NOVEL JOINT OPTIMIZATION OF GRADIENT BOOSTING DECISION TREES AND SARIMAX MODELS FOR NONLINEAR TIME SERIES REGRESSION USING A STATE SPACE APPROACH

A THESIS SUBMITTED TO

THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF

MASTER OF SCIENCE

IN

ELECTRICAL AND ELECTRONICS ENGINEERING

By

Ahmet Berker KOÇ

August 2025

NOVEL JOINT OPTIMIZATION OF GRADIENT BOOSTING DECISION TREES AND SARIMAX MODELS FOR NONLINEAR TIME SERIES REGRESSION USING A STATE SPACE APPROACH

By Ahmet Berker KOÇ

August 2025

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Süleyman Serdar Kozat(Advisor)

Tolga Çukur

Ramazan Gökberk Cinbiş

Approved for the Graduate School of Engineering and Science:

Orhan Arıkan
Director of the Graduate School

ABSTRACT

NOVEL JOINT OPTIMIZATION OF GRADIENT BOOSTING DECISION TREES AND SARIMAX MODELS FOR NONLINEAR TIME SERIES REGRESSION USING A STATE SPACE APPROACH

Ahmet Berker KOÇ
M.S. in Electrical and Electronics Engineering
Advisor: Süleyman Serdar Kozat
August 2025

We investigate nonlinear regression or forecasting in an online setting and propose an end-to-end ensemble model that addresses two key challenges: the trade-off between linear and nonlinear modeling, and the disjoint nature of the optimization process. The proposed architecture seamlessly integrates a linear time series model and a nonlinear soft decision tree-based model within a unified structure. Specifically, the architecture employs a Seasonal Autoregressive Integrated Moving Average with Exogenous Regressors (SARIMAX) model to capture key linear patterns in time series data, such as seasonality and trend, alongside a soft gradient boosting decision tree (SGBDT), which is adept at modeling nonlinear dependencies and extracting features directly from raw inputs. Different from existing state-of-the-art hybrid approaches that typically rely on sequential and disjoint training strategies, for the first time in the literature, we introduce a jointly optimized hybrid model in which both the linear and nonlinear components are trained simultaneously. This is achieved through a novel formulation based on state-space representations, which allows the integration of the ARMA-based SARIMAX model and the SGBDT into a single state-space framework. We employ the Extended Kalman Filter (EKF), a nonlinear optimization technique, to efficiently perform the joint training process by leveraging derived state transition and measurement equations. Furthermore, the architecture is modular and extensible, allowing for the substitution or addition of alternative nonlinear models and other linear time series techniques, such as Exponential Smoothing (ETS), as long as state representations are obtained. The optimization component is also flexible, enabling the replacement of the EKF with other techniques such as the Unscented Kalman Filter (UKF) or Particle Filters (PF). Through extensive experiments on well-known real-world datasets, the proposed approach

demonstrates superior performance. We provide the source code as a publicly available repository to support further research and ensure reproducibility.

Keywords: time series, state space models, online learning, ensemble learning, prediction/regression, soft decision trees, gradient boosting, soft gradient boosting trees, seasonal autoregressive integrated moving average with exogenous regressors (SARIMAX), nonlinear optimization, extended Kalman filter (EKF).

ÖZET

DURUM UZAYI YAKLAŞIMI KULLANILARAK DOĞRUSAL OLMAYAN ZAMAN SERİSİ REGRESYONU İÇİN GRADYAN ARTIRICI KARAR AĞAÇLARI VE SARIMAX MODELLERİNİN YENİ BİRLEŞİK OPTİMİZASYONU

Ahmet Berker KOÇ
Elektrik ve Elektronik Mühendisliği, Yüksek Lisans
Tez Danışmanı: Süleyman Serdar Kozat
Ağustos 2025

Çevrimiçi bir ortamda doğrusal olmayan regresyon veya tahminlemeyi araştırıyoruz ve doğrusal ve doğrusal olmayan modelleme arasındaki denge ve optimizasyon sürecinin kopuk doğası gibi iki temel zorluğu ele alan uçtan uca bir topluluk modeli öneriyoruz. Önerilen mimari, doğrusal bir zaman serisi modelini ve doğrusal olmayan bir yumuşak karar ağacı tabanlı modeli birleşik bir yapı içinde sorunsuz bir şekilde entegre ediyor. Özellikle, mimari, mevsimsellik ve trend gibi zaman serisi verilerindeki temel doğrusal örüntüleri yakalamak için Dışsal Regresörlü Mevsimsel Otoregresif Entegre Hareketli Ortalama (SARIMAX) modelini ve doğrusal olmayan bağımlılıkları modellemede ve özellikleri doğrudan ham girdilerden çıkarmada usta olan yumuşak gradyan artırıcı karar ağacını (SGBDT) kullanıyor. Genellikle sıralı ve kopuk eğitim stratejilerine dayanan mevcut en son hibrit yaklaşımlardan farklı olarak, literatürde ilk kez, hem doğrusal hem de doğrusal olmayan bileşenlerin aynı anda eğitildiği ortak olarak optimize edilmiş bir hibrit model sunuyoruz. Bu, ARMA tabanlı SARIMAX modelinin ve SGBDT'nin tek bir durum-uzay çerçevesine entegre edilmesini sağlayan, durum-uzay temsillerine dayalı yeni bir formülasyon aracılığıyla elde edilir. Türetilmiş durum geçişi ve ölçüm denklemlerinden yararlanarak ortak eğitim sürecini verimli bir şekilde gerçekleştirmek için doğrusal olmayan bir optimizasyon tekniği olan Genişletilmiş Kalman Filtresi'ni (EKF) kullanıyoruz. Ayrıca, mimari modüler ve genişletilebilir olup, durum temsilleri elde edildiği sürece alternatif doğrusal olmayan modellerin ve Üstel Düzeltme (ETS) gibi diğer doğrusal zaman serisi tekniklerinin değiştirilmesine veya eklenmesine olanak tanır. Optimizasyon bileşeni de esnektir ve EKF'nin Kokusuz Kalman Filtresi (UKF)

veya Parçacık Filtreleri (PF) gibi diğer tekniklerle değiştirilmesini sağlar. İyi bilinen gerçek dünya veri kümeleri üzerinde yapılan kapsamlı deneyler sonucunda, önerilen yaklaşım üstün bir performans göstermiştir. Daha fazla araştırmayı desteklemek ve tekrarlanabilirliği sağlamak için kaynak kodu açık erişimli biçimde paylaşıyoruz.

Anahtar sözcükler: zaman serisi, durum uzayı, çevrimiçi öğrenme, topluluk öğrenmesi, tahmin/regresyon, yumuşak karar ağaçları, gradyan artırma, yumuşak gradyan artırma ağaçları, dışsal regresörlere sahip mevsimsel otoregresif entegre hareketli ortalama (SARIMAX), doğrusal olmayan optimizasyon, genişletilmiş Kalman filtresi (EKF).

Acknowledgement

First and foremost, I would like to express my deepest gratitude to my advisor, Prof. Süleyman Serdar Kozat, for their invaluable guidance, continuous support, and encouragement throughout my master's degree. His expertise, insightful feedback, and patience helped me to improve theoretical and practical aspects. I feel truly privileged to have had the opportunity to learn under their supervision.

I would like to express my sincere appreciation to my family for their steadfast support throughout my studies. In particular, I am deeply grateful to my wife for her patience, understanding, and constant encouragement, which have been a pillar of strength for me. I also extend my heartfelt thanks to my mother and father for their enduring belief in my potential and the many sacrifices they have made to support my academic pursuits. I am thankful to my sister and brother for their continued motivation and thoughtful presence. Additionally, I am grateful to my grandmother for praying for me before every exam. I would also like to acknowledge the support of my extended family and relatives for their kind words and moral support. Their collective support has been vital to the completion of my master's journey.

I would also like to extend my sincere thanks to my friends and colleagues who have been an integral part of this journey. I am particularly grateful to my research team—Arda Fazla, Selim Furkan Tekin, Furkan Burak Mutlu, Emirhan İlhan, Efe Lorasdağı, Mustafa Enes Aydın, Doğan Can Çiçek, İsmail Balaban, Mehmet Yiğit Turalı, Ramazan Burak Güler, and Hüseyin Karaca—for their collaboration, insightful discussions, and invaluable contributions to our shared work. I am also thankful to my close friends Bilal Onur Eskili, Burak Akgül, Mustafa Mert Keskin, Ali İlker Sığırcı, İsmail Hakkı Yıldız, Muhammed Yusuf Aksel, Ömer Faruk Arslan, Sefa Karaca, Buğrahan Kara, and Oğuzhan Şahin for their continuous encouragement, camaraderie, and support, which have greatly enriched my academic experience and personal well-being throughout this process.

I would also like to thank Türk Telekom A.Ş. for their support with the 5G

and Beyond Graduate Scholarship Programme. Their contribution has not only provided vital financial assistance during my graduate studies but has also served as a source of motivation and encouragement in pursuing advanced research.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

We investigate the problem of nonlinear prediction and regression within the context of time series analysis, where we receive a noisy data sequence associated with the target signal and estimate the next sample of the sequence. This problem has been widely explored in both the signal processing and machine learning communities due to its broad range of real-world applications, including traffic flow analysis [1], meteorological studies [2], energy demand forecasting [3], electric load forecasting [4], web service workload prediction [5], and financial market modeling [6].

One of the fundamental challenges in time series forecasting lies in selecting between linear and nonlinear modeling approaches. Linear models are frequently employed due to their conceptual simplicity, computational efficiency, and capacity to produce satisfactory results in many scenarios. There exist linear models that have been developed explicitly for time series tasks, such as the autoregressive moving average (ARMA) family. For instance, SARIMAX (seasonal autoregressive integrated moving average with exogenous variables) from the ARMA family explicitly models seasonality, while ETS (exponential smoothing) incorporates trend and seasonal components directly into its structure. Nevertheless, these models rely on strong assumptions, particularly linearity and stationarity, that limit their ability to capture complex, nonlinear patterns commonly present

in real-world time series data [7]. In other words, although these models are designed for time series prediction, their linear nature restricts their scalability and forecasting performance [8]. To overcome these limitations, nonlinear models such as decision tree-based algorithms, recurrent neural networks (RNNs), and transformer-based architectures are commonly used for real-world time series problems. These models not only account for nonlinear relationships but also theoretically subsume linear models. They offer at least comparable or even superior predictive performance in principle. However, the enhanced representational capacity of nonlinear models comes at the cost of increased optimization complexity and greater data requirements for effective training. As a result, hybrid approaches that combine both linear and nonlinear models have gained popularity in the signal processing and machine learning literature because they aim to balance model flexibility with robustness. Despite their advantages, such hybrid approaches often face performance bottlenecks when the constituent models are combined in a disjoint manner. In such cases, the final output is limited by the best-performing model or adversely affected by weaker components in the ensemble.

Another significant challenge in time series forecasting lies in the disjointed nature of the optimization process. Traditionally, this disjointness arises from a two-stage process: domain experts manually extract relevant features, which are subsequently input into an independent predictive model. As shown in our experiments, this disjoint optimization often leads to sub-optimal results because the handcrafted features may not align well with the underlying learning algorithm, thereby limiting its ability to exploit data effectively. To address optimization challenges and enhance predictive performance, researchers have investigated alternative strategies. Two of the most prominent methods are ensemble-based techniques such as gradient boosting machines and deep learning approaches. There are attempts to overcome the disjointness problem by combining objectives in a single framework. One of the most important developments here has been the introduction of gradient boosting machines (especially decision tree-based ones [9, 10]) to the field of time series forecasting. GBDTs combine multiple

weak learners to build a powerful ensemble capable of modeling complex nonlinear relationships between input features and target sequences. However, these models are prone to overfitting, often exhibiting high variance. Particularly in time series forecasting, GBDTs tend to overemphasize the most recent observations, and then they behave similarly to naive forecasting models [11]. On the other hand, from the raw data, deep learning models extract features inherently. This capability enables them to combine feature extraction and the training process seamlessly. That is, they simultaneously learn informative representations and fit the model to the data within a single framework. Recent developments in transformer-based architectures, inspired by their success in natural language processing and computer vision, have led to specialized adaptations for time series forecasting [12], [13], [14]. These models excel at capturing long-range dependencies, multi-scale temporal patterns, and complex relationships; however, their performance is heavily dependent on access to a large amount of training data to reach effective performance. Since time series datasets are often relatively limited in size, this constraint presents a significant challenge for such models. While time series data can span long durations, the temporal patterns they exhibit are typically non-stationary and continuously evolving. Consequently, relying on distant historical data may lead to inaccurate representations of present or future trends, which can negatively impact the model's ability to learn effectively. Therefore, for the limited data, deep learning models are prone to overfitting and likely to exhibit suboptimal performance. As a result, both gradient boosting and deep learning models tend to converge to simple or near-linear solutions [15], primarily focusing on a single dominant component such as the prominent seasonal patterns or lag values typically found in time series data, since these features are relatively straightforward for the model architectures to capture. However, this behavior results in a significant degradation of model performance, as the more important nonlinear relationships in the data are not captured effectively, e.g., a LightGBM [16] easily converges to a (seasonal) naive model as we show in our experiments.

We propose a unified, end-to-end algorithm that integrates both feature selection and model training by combining the strengths of linear and nonlinear

modeling approaches within a joint architecture. Specifically, we employ a soft gradient boosting decision tree (SGBDT) to capture complex nonlinear patterns and perform automatic feature extraction, while leveraging a SARIMAX model to model linear components such as trend and seasonality effectively. Exogenous variables are processed through nonlinear transformations in the SGBDT and through linear mappings in SARIMAX, enabling a complementary "separation of duties" between the models. The two components are jointly optimized within a unified nonlinear state-space formulation, using methods such as the Extended Kalman Filter (EKF). This framework is flexible and extensible, allowing substitution of SARIMAX with other classical time series models (e.g., ETS), or replacement of SGBDT with other boostable models that can be formulated within a state-space form, as well as alternative filtering techniques like the Unscented Kalman Filter (UKF) or Particle Filter (PF).

In Chapter 2, we introduce a unified framework to ensemble an SGBDT and a SARIMAX model. Specifically, we use a soft gradient boosting decision tree (SGBDT) to capture complex nonlinear patterns and perform automatic feature extraction, while employing a SARIMAX model to capture linear patterns, namely trend and seasonality effectively. We derive a novel state-space representation for both SGBDT and SARIMAX. Then, we can integrate them into a single, coherent state-space formulation. Joint optimization of the hybrid model is performed using the EKF, a nonlinear state estimation technique, applied to the unified state space. This framework allows us to leverage the complementary strengths of time series-specific linear modeling and nonlinear representation learning through soft decision trees and the gradient boosting algorithm. The main contributions of Chapter 2 are as follows:

1. We propose a unified architecture that integrates a soft gradient boosting machine with a SARIMAX model for online or sequential time series prediction. This architecture is designed to be trainable in an end-to-end manner by using the Extended Kalman Filter (EKF). To the best of our knowledge, this is the first model in the literature that enables joint optimization of a nonlinear soft tree-based component and a conventional linear time series

predictor.

2. We present state-space formulations that support efficient one-pass parameter estimation, thereby enabling online learning and the modular concatenation of multiple state spaces.

3. The proposed architecture is highly flexible: the nonlinear component can be substituted with any other differentiable machine learning models if state representation can be obtained, while the linear component may be replaced with alternative linear time series models, such as ETS [17]. Furthermore, end-to-end training is not limited to EKF and may also be achieved via alternative state-space optimization techniques, such as PF and UKF.

4. We demonstrate the effectiveness of the proposed model through extensive experiments on well-known benchmark datasets. We achieve competitive or superior performance compared to state-of-the-art methods.

5. To promote reproducibility and support future research, we publicly release the implementation of our model[1]

Notation: All vectors are denoted using boldface lowercase letters, while matrices are represented by boldface uppercase letters. For instance, $\boldsymbol{x}$ denotes a vector whereas $\boldsymbol{X}$ denotes a matrix. Each entry of the vectors and matrices is a real number. The transpose of a matrix is indicated by the superscript T, as in $\boldsymbol{X}^T$, and the inverse of a matrix is denoted by the superscript -1, as in $\boldsymbol{X}^{-1}$. Furthermore, a slice of a vector is expressed as $\boldsymbol{x}_{a:b}$, where a is inclusive and b is exclusive, resulting in a length of $b-a$. Additionally, we use $\boldsymbol{x}_{t,i}$, where t represents the time step and i denotes the index of the vector $\boldsymbol{x}$.

[1] https://github.com/ahmetberkerkoc/SARIMAX_SGBDT_with_EKF

Chapter 2

Joint Optimization of Gradient Boosting Decision Trees and SARIMAX Models in a Unified State Space

In this work, we propose an end-to-end algorithm that simultaneously performs feature selection and model training, seamlessly integrating both linear and nonlinear components within a unified architecture. Specifically, we employ a *soft* GBDT (SGBDT) [18] to enable automatic feature extraction at its nodes and to model nonlinear patterns effectively. Also, at the same time, we incorporate a time series-oriented linear model, SARIMAX, which is well suited for capturing trend and seasonality dynamics. Furthermore, exogenous variables (i.e., side information) undergo various nonlinear transformations within the SGBDT to contribute to the prediction task, whereas the SARIMAX model leverages a direct linear mapping between these variables and the target signal to produce the final prediction. Therefore, we construct an effective ensemble by integrating an SGBDT and a SARIMAX model in parallel, leveraging a "separation of duties" strategy to emphasize their respective strengths, while simultaneously performing joint optimization of the overall model parameters. To this end, we formulate a

unified nonlinear state-space model and optimize the entire system in an end-to-end fashion using, for instance, EKF. Importantly, the proposed framework is modular: alternative linear time series models (e.g., ETS) can replace SARIMAX, other nonlinear models with state-space representations can substitute SGBDT, and different state estimation methods such as UKF or PF can be employed for optimization.

2.1 Background

2.1.1 Literature Review

In this section, we provide a comprehensive review of time series forecasting methods, covering statistical models, machine learning approaches, including classical techniques, tree-based methods, and deep learning architectures. In addition, we explain the optimization methods used for state estimation. This review highlights the evolution of forecasting techniques and presents a wide range of models used in the signal processing and machine learning literature so that we can provide essential context for the development of our proposed unified architecture.

2.1.1.1 Statistical Models

Linear statistical models are widely used in time series forecasting and have formed a basis due to their mathematical simplicity and analytical tractability. These models assume that the future values of a time series can be expressed as linear combinations of its past values and past error terms. Despite their simplicity, linear models are highly effective in a wide range of applications. In particular, when the underlying time series is stationary or can be transformed into a stationary process through differencing, statistical models can yield significant performance.

2.1.1.1.1 ARMA-based Models

In statistical models for time series forecasting, ARMA-based models are among the most commonly employed techniques. The models have actually become more complex by building upon each other. The simplest and most understandable of them is the Auto-Regressive (AR) model. It uses a linear combination of a fixed number of past observations to generate a prediction for the current value. The number of past observations included in the model is a hyperparameter known as the order and is denoted by p. It is formally defined as

$$\begin{aligned} y_t &= c + \alpha_1 y_{t-1} + \alpha_2 y_{t-2} + \ldots + \alpha_i y_{t-i} + \epsilon_t \\ &= c + \sum_{i=1}^{p} \alpha_i y_{t-i} + \epsilon_t, \end{aligned} \tag{2.1}$$

where c represents a constant term, ϵ_t is the white noise and the coefficient α_i denotes the ith parameter of the related auto-regressive variable. Then, the Moving Average (MA) model defines the current observations in terms of a linear function of the past error terms, given as

$$\begin{aligned} y_t &= c + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \ldots + \theta_i \epsilon_{t-i} + \epsilon_t \\ &= c + \sum_{i=1}^{q} \theta_i \epsilon_{t-i} + \epsilon_t, \end{aligned} \tag{2.2}$$

where the coefficient θ_i denotes the ith parameter of the related moving average variable and q represents the number of past errors used in the model. Both models provide valuable insights, but have limitations when applied independently. To address these limitations, the Auto-Regressive Moving Average (ARMA) model combines the strengths of AR and MA components into a single equation so that the model can capture both the dependency of past observations and errors, given as

$$y_t = c + \sum_{i=1}^{p} \alpha_i y_{t-i} + \sum_{i=1}^{q} \theta_i \epsilon_{t-i} + \epsilon_t. \tag{2.3}$$

These models are based on the strong assumption that the data is stationary. However, most real-life time series data are non-stationary, meaning that their statistical properties change over time. To handle non-stationary data, the Auto-Regressive Integrated Moving Average (ARIMA) model is developed. It introduces a differencing operation to transform non-stationary data into stationary data before applying the ARMA model. In other words, ARIMA is structurally equivalent to ARMA, but the target is the differenced form of the actual series. The first-order differencing is calculated as

$$y_t^{'} = y_t - y_{t-1}. \tag{2.4}$$

Since the first observation has no prior value for differencing, the differenced series contains one fewer data point than the original series. The differencing order is controlled by a hyperparameter d. Yet, ARIMA models assume that the time series does not have seasonal dynamics, which are commonly observed in practical forecasting scenarios. To capture both short-term and seasonal patterns in time series data, the Seasonal ARIMA (SARIMA) model extends ARIMA by incorporating seasonal components. In addition to the non-seasonal orders (p, d, q) from ARIMA, SARIMA introduces seasonal orders (P, D, Q, s), where (P, D, Q) are the seasonal counterparts of the autoregressive, differencing, and moving average terms, respectively. The parameter s represents the length of the seasonal cycle, for example, 12 for monthly data with yearly seasonality. The extended SARIMA formulation is thus expressed as:

$$y_t = c + \sum_{i=1}^{p} \alpha_i y_{t-i} + \sum_{i=1}^{q} \theta_i \epsilon_{t-i} + \sum_{i=1}^{P} \phi_i y_{t-si} + \sum_{i=1}^{Q} \eta_i \epsilon_{t-si} + \epsilon_t. \tag{2.5}$$

In many applications, the target time series is influenced not only by its own past values but also by exogenous variables. Therefore, the SARIMAX model addresses this by incorporating exogenous variables. It employs the order notation of SARIMA while adding a linear regression component for the exogenous variables, given as

$$y_t = c + \sum_{i=1}^{p} \alpha_i y_{t-i} + \sum_{i=1}^{q} \theta_i \epsilon_{t-i} + \sum_{i=1}^{P} \phi_i y_{t-si} + \sum_{i=1}^{Q} \eta_i \epsilon_{t-si} + \sum_{i=1}^{X} \kappa_i \boldsymbol{x}_{t,i} + \epsilon_t \quad (2.6)$$

Taking exogenous variables into account enables us to obtain more context-aware forecasts and improves prediction accuracy. ARMA-based models offer a powerful framework for modeling linear relationships in time series data, particularly when the series is stationary or can be transformed into a stationary series by differencing. Their ability to incorporate both auto-regressive and moving average components enables them to capture short-term temporal dependencies effectively. Moreover, with the inclusion of seasonal terms and exogenous variables in SARIMA and SARIMAX extensions, these models are adapted to a wide range of forecasting applications.

As a result, ARMA-based models are well-suited for capturing linear dependencies in stationary and seasonally adjusted time series. They are especially effective in stationary data, or we can apply differencing to obtain it. With the addition of exogenous variables, it provides a complete framework for time series.

2.1.1.1.2 Exponential Smoothing Models

Another statistical model family is the exponential smoothing. These models are suitable for forecasting when the data have no clear trend or seasonal pattern. ETS models are constructed based on the principles of smoothing. To understand the foundations of ETS, it is useful to begin with basic forecasting strategies that illustrate the underlying logic behind the ETS. The naive method uses the most recent observation as the prediction, formally expressed as $\hat{y}_{t+1} = y_t$. It assumes that the last value is the most important and the only feature. Another method is the average method, which forecasts by giving equal weight to a simple average of the observed data, given as

$$\hat{y}_{T+1} = \frac{1}{T} \sum_{i=1}^{T} y_i. \quad (2.7)$$

However, this method assumes that all past values have the same importance by giving equal weight. To address the limitations of both the naive and average

models, there is another model that applies exponentially decaying weights to past observations, assigning greater importance to more recent observations, given as

$$\hat{y}_{T+1} = \sum_{k=1}^{T} \alpha(1-\alpha)^{k-1} y_{T-k+1}, \tag{2.8}$$

where $\alpha \in [0, 1]$ denotes the smoothing parameter that controls the rate of exponential decay. Simple exponential smoothing (SES) is based exactly on this model. SES is computed as a weighted average between the last observation and the previously forecast value. Formally, the forecast at time $t+1$ is given by

$$\hat{y}_{t+1} = \alpha y_t + (1-\alpha)\hat{y}_t, \tag{2.9}$$

where y_t is the actual observation at t, $\hat{y}_t$ is the forecast value for t, and $\alpha \in [0, 1]$ is the smoothing parameter. Hence, if we write the formula recursively from time $t = 1$, we obtain exactly the same equation as (2.8). Next, Holt's linear trend method extends Simple Exponential Smoothing to handle time series data with a linear trend component. To this end, this method introduces two smoothing equations: the first is the level equation, and the second is the trend equation. The model is defined by

$$\begin{aligned} l_t &= \alpha y_t + (1-\alpha)(l_{t-1} + b_{t-1}) \\ b_t &= \beta(l_t - l_{t-1}) + (1-\beta) b_{t-1} \end{aligned} \tag{2.10}$$

where l_t and b_t represent the estimated level and trend at time t, respectively, while α and β are the corresponding smoothing parameters, both bounded within the range $[0, 1]$. Then, the forecast is computed as

$$\hat{y}_{t+h} = l_t + h b_t \tag{2.11}$$

where h denotes the number of time steps ahead to forecast. For example, $h = 1$ is used for one-step-ahead prediction.

Building upon Holt's linear trend model, the Holt-Winters methods introduce a seasonal component, making them applicable for time series data that exhibit both trend and seasonality. There are two main variants: additive and multiplicative, chosen based on the nature of the seasonal fluctuations. The additive

method assumes that seasonal effects are approximately constant over time and adds them to the trend and level components. In contrast, the multiplicative method assumes that seasonal variations change proportionally with the level of the series; and therefore, multiplies the seasonal component with the trend-adjusted level as (2.11). As a result, compared to the ARMA-based models, when the data exhibit more dynamic patterns or require adaptive forecasting, exponential smoothing models offer a more suitable alternative. By assigning exponentially decreasing weights to past observations, ETS models can adapt more responsively to recent changes. These methods can model level, trend, and seasonality using additive or multiplicative components. This flexibility makes them effective for forecasting non-stationary time series without the need for explicit differencing.

From simple auto-regressive or naive structures to more advanced models, namely SARIMAX and ETS, these approaches provide interpretable mechanisms to capture both short-term dynamics and seasonal patterns. Their modular nature allows for easy integration of exogenous variables, differencing techniques, and smoothing mechanisms adapted to the characteristics of the data. Despite the rise of machine learning and deep learning methods, statistical models remain widely used due to their interpretability, low data requirements, and effectiveness in a wide range of forecasting applications.

2.1.1.2 Machine Learning Models

2.1.1.2.1 Tree-Based Models

The Decision Tree is a classical supervised learning algorithm structured as a hierarchical tree composed of a root node, internal (decision) nodes, leaf nodes, and connecting branches. It recursively splits the input space based on feature thresholds to predict target values. This process forms a series of axis-aligned partitions that enable the model to capture nonlinear relationships. Each internal node applies a hard decision rule, meaning that a single branch is deterministically selected based on whether the selected input feature satisfies a specific condition in the internal node. This process continues until a leaf node is reached, which

assigns a class label (classification) or a predicted value (regression). Due to its splitting mechanism, the decision tree algorithm does not need any feature scaling or prior data transformation.

Another decision tree model is the Soft Decision Tree. Different from the classical (hard) decision tree, it introduces differentiability by replacing hard splits with probabilistic decisions [19]. In the internal nodes, the soft decision tree routes the input to both child nodes with probabilities that depend on the input features. As a result, all paths from the root to the leaf nodes are traversed, and each leaf node contributes to the final prediction weighted by the probability of its corresponding path. Hence, the model to learn smooth decision boundaries in the input space, rather than relying only on axis-aligned partitions.

Individual trees, such as hard and soft decision trees, suffer from high variance or limited predictive power. To address these limitations, ensemble methods such as Random Forests and Gradient Boosted Decision Trees combine multiple trees to improve accuracy and generalization [20]. Random forest is an ensemble learning algorithm based on the bagging technique. It constructs a large number of decision trees during training and aggregates their outputs to produce a final prediction, i.e., majority voting for classification and averaging for the regression. Each tree is trained on a random subset of the training set, and the feature set is also randomly selected at each split. This randomness in both data and feature selection introduces diversity among the individual trees, which helps to reduce overfitting and improve generalization capability. It is important to note that due to the independence of tree construction, the Random Forest algorithm supports efficient parallel computation.

In contrast, Gradient Boosted Decision Trees (GBDTs) are based on the boosting technique, where trees are built sequentially, rather than independently. In this algorithm, multiple weak learners are combined sequentially to construct a strong learner. In the case of GBDTs, each weak learner is a decision tree, and each decision tree is trained to correct the residual errors made by the ensemble of previous trees. GBDTs have demonstrated outstanding performance across a

wide range of real-world applications. XGBoost [21], LightGBM [16], and Catboost [22] are the most popular GBDT frameworks. They are highly effective for time series forecasting when combined with appropriate feature engineering techniques, such as lags, trends, or seasonal indicators. Moreover, they can be trained using a variety of loss functions, provided that the functions are differentiable.

2.1.1.2.2 Classical ML Models

Several classical machine learning models have been applied to time series forecasting [23]. These models are particularly effective when the input data includes exogenous features and lag-based features. Support Vector Regression (SVR) is widely used because it models nonlinear relationships through kernel functions while maintaining strong generalization. k-Nearest Neighbors (kNN) regression provides a straightforward approach. It does not need any training. It just measures the distance of the test data and all instances in the dataset. Then, for a given input, the algorithm identifies the k most similar instances, typically using Euclidean distance, and computes the prediction as the average of their corresponding target values. Regularized linear models, such as Ridge Regressor and Lasso, are often used in high-dimensional feature spaces to prevent overfitting and improve generalization. These models extend the classical linear regressor by modifying the ordinary least squares (OLS) objective function. It adds a penalty term to the loss function to constrain the size of the model coefficients. Ridge Regressor penalizes the sum of squared coefficients, whereas Lasso penalizes the sum of absolute coefficients, given as

$$\begin{aligned}
&\textbf{Classical Linear Regressor:} && \min_{\boldsymbol{\beta}} \sum_{i=1}^{n} \left(y_i - \boldsymbol{x}_i^\top \boldsymbol{\beta}\right)^2 \\
&\textbf{Ridge Regressor:} && \min_{\boldsymbol{\beta}} \sum_{i=1}^{n} \left(y_i - \boldsymbol{x}_i^\top \boldsymbol{\beta}\right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \\
&\textbf{Lasso Regressor:} && \min_{\boldsymbol{\beta}} \sum_{i=1}^{n} \left(y_i - \boldsymbol{x}_i^\top \boldsymbol{\beta}\right)^2 + \lambda \sum_{j=1}^{p} |\beta_j|
\end{aligned} \tag{2.12}$$

where $\boldsymbol{x}_i$ and y_i represent input vector and target value for sample i, respectively,

$\boldsymbol{\beta}$ denotes the coefficient vector to be learned and λ is regularization parameter. These models are particularly useful when a large number of features are included in the forecasting model. They are effective in reducing overfitting and ensuring consistent performance.

2.1.1.2.3 Deep Learning Models

Artificial Neural Networks (ANNs) are composed of layers of interconnected neurons. These layers are the input layer, hidden layers, and output layer. Each layer applies a nonlinear transformation to the input by using an activation function such as ReLU [24], sigmoid, and tanh. ANNs are commonly employed to approximate a broad class of nonlinear functions across various domains [25]. In time series data, an ANN is fed with targets and corresponding features, and it can model complex nonlinear relationships [26]. However, standard ANNs do not have a memory mechanism, i.e., they treat each input independently. Hence, this limits their use for sequential or time-dependent data.

Recurrent Neural Networks (RNNs) address the need for memory in neural networks, especially for sequential data. There is a loop in the RNN architecture so that information can be kept between time steps. At each step in a sequence, an RNN takes both the current input and the previous hidden state to compute the new hidden state. This enables the model to adapt to temporal dynamics. However, basic RNNs struggle to learn long-term dependencies. The vanishing or exploding gradient problems can occur when sequences become long.

Long Short-Term Memory networks are an advanced version of RNNs. Especially, they are designed to overcome limitations of RNNs in remembering long-term dependencies [27]. LSTMs have a more complex architecture that includes memory cells and gating mechanisms such as the input, forget, and output gates. They control how information is stored, forgotten, or passed along over time. In this way, LSTMs can remember relevant context over many time steps, where standard RNNs typically forget. This structure makes them highly effective for sequential tasks, namely time series forecasting, language modeling, and machine translation. In addition to LSTM, there is another advanced version of RNNs

named the Gated Recurrent Unit (GRU). GRUs simplify the LSTMs by combining the forget and input gates into a single update gate. They generally achieve similar performance to LSTMs while offering less computational cost due to their reduced complexity.

In recent years, new time series forecasting models have been developed. With the advent of Transformers, particularly in domains such as natural language processing, they have recently been extended to the domain of time series forecasting. The main problem of the transformer-based models is the lack of data to train a complex transformer architecture. Also, traditional transformers struggle with time series due to their time and memory complexity with respect to sequence length. This makes them impractical for long inputs. To address this issue, Informer [12] introduces a novel ProbSparse self-attention mechanism. This mechanism selects only the most important query-key pairs instead of computing full attention across all positions. It also incorporates a self-attention distilling layer that compresses long input sequences into shorter and informative representations. Another transformer-based model, iTransformer [13], focuses on improving multivariate forecasting tasks. Different from the traditional transformers, which embed the time steps, iTransformer embeds features as separate tokens so that it can model the relationships between different features and their impact on the target value. It also introduces a learnable instance normalization layer in the frequency domain, which stabilizes training and adapts to varying time series scales.

Another advanced deep learning forecasting model is DeepAR, developed by Amazon [28]. It uses an auto-regressive recurrent neural network (RNN), i.e., a combination of past observations is used to produce predictions for each time step. In contrast to conventional methods that produce deterministic predictions, it predicts a probability distribution that models the distributions associated with future values. Moreover, several advanced deep learning models have been proposed for time series forecasting, including Fedformer [14], Autoformer [29], TimeMixer [30], TsMixer [15], and the Temporal Fusion Transformer (TFT) [31].

2.1.1.3 Optimization and State Estimation

2.1.1.3.1 Kalman Filter

The Kalman Filter is a recursive algorithm that estimates the state of a dynamic system [32], i.e., system parameters, by using a sequence of noisy and possibly incomplete measurements. It has two main steps, which are the prediction and the update. In the prediction step, the filter estimates the current state and its uncertainty using a known system model. The inputs to this model are the previous state and available control inputs. The prediction step includes two key components. The first component is the state prediction, where the future state is estimated based on the previous state, given by

$$\boldsymbol{z}_t = \boldsymbol{G_t z_{t-1|t-1}} + \boldsymbol{B_t u_t}, \tag{2.13}$$

where $\boldsymbol{z}_t$ is the predicted state, $\boldsymbol{G_t}$ denotes the state transition matrix to change state without control input, $\boldsymbol{B_t}$ represents the control input matrix to map the control input to state space, $\boldsymbol{u_t}$ is the control input vector. Next, the second key component of the prediction step is the state covariance prediction, where the uncertainty of this prediction is computed as

$$\boldsymbol{Z_{t|t-1}} = \boldsymbol{G_t Z_{t-1|t-1} G_t^T} + \boldsymbol{C_t}, \tag{2.14}$$

where $\boldsymbol{Z_{t|t-1}}$ represents the predicted covariance and $\boldsymbol{C_t}$ is the noise covariance. Based on the current state and known system, the algorithm generates a prediction for measurement, given as

$$\hat{y}_t = \boldsymbol{P_t z}_t + \hat{\epsilon}_t, \tag{2.15}$$

where $\boldsymbol{P_t}$ maps the current state to the prediction for the measurement, and $\hat{\epsilon}_t$ denotes the white noise process. Then, the residual or error is the difference between the actual and predicted measurement and is calculated as $r_t = y_t - \hat{y}_t$. In the update step, this prediction is corrected to decrease the error by using measurements (i.e., observations) and their uncertainties, which are computed by

a quantity known as the Kalman Gain. The Kalman Gain is computed as

$$\begin{aligned} \boldsymbol{K_t} &= \boldsymbol{Z_{t|t-1}} \boldsymbol{P_t^T} \boldsymbol{R_t^{-1}} \\ \boldsymbol{R_t} &= \boldsymbol{P_t} \boldsymbol{Z_{t|t-1}} \boldsymbol{P_t} + \boldsymbol{Q_t} \end{aligned} \tag{2.16}$$

where $\boldsymbol{Q_t}$ represents the noise covariance matrix. Then, the predicted state and the predicted state covariance are corrected by using the new measurement and the calculated Kalman Gain, given as

$$\begin{aligned} \boldsymbol{z_{t|t}} &= \boldsymbol{z}_t + \boldsymbol{K_t} \boldsymbol{r_t} \\ \boldsymbol{Z_{t|t}} &= (\boldsymbol{I} - \boldsymbol{K_t} \boldsymbol{P_t}) \boldsymbol{Z_{t|t-1}} \end{aligned} \tag{2.17}$$

where $\boldsymbol{r_t}$ is the 1×1 matrix form of the residual r_t.

The Kalman Filter is based on the assumption that the system can be modeled as a linear dynamical process with Gaussian noise, and it aims to minimize the mean squared error of the state prediction. Specifically, it assumes that the system follows a linear dynamical model, where the state transitions are described by linear equations, and observations are linear functions of the state. These linear equations are formally known as the process and measurement equations. Both the process and the measurement are affected by noise, which is assumed to be zero-mean Gaussian with known covariance matrices. This probabilistic structure allows the Kalman Filter to optimally estimate the system state by combining model predictions and observed data, taking their respective uncertainties into account. Due to its efficiency and accuracy under these assumptions, the Kalman Filter has become a fundamental algorithm in control systems, robotics, and signal processing [33].

2.1.1.3.2 Extended Kalman Filter

The Extended Kalman Filter (EKF) is developed to handle nonlinear systems, which the standard Kalman Filter can not solve citefujii2013extended. It generalizes the Kalman Filter to nonlinear systems. In the EKF, state prediction and measurement prediction are represented by nonlinear functions such as $\varrho(\cdot)$ and $\rho(\cdot)$. Then, it performs first-order Taylor expansions to approximate these nonlinear functions. After Taylor expansions, Jacobian matrices are obtained to

linearize the system. The prediction state turns out $\boldsymbol{z}_t = \varrho(\boldsymbol{z}_{t-1}, \boldsymbol{u_t})$ from (2.13), and the Jacobian of the process model $\boldsymbol{G_t} = \frac{\partial \varrho}{\partial \boldsymbol{z}}$ is used instead of the linear state transition matrix in (2.14). Similarly, the update step computes the predicted measurement as $\hat{y}_t = \rho(z_t)$, and Jacobian of the measurement model $\boldsymbol{P_t} = \frac{\partial \rho}{\partial \boldsymbol{z}}$ is used instead of $\boldsymbol{P_t}$ to linearize measurement model. Therefore, the Kalman Gain and all other formulations in standard Kalman Gain are adapted to nonlinear systems by replacing the constant matrices with Jacobians of the process and measurement models. Thus, the Extended Kalman Filter extends the standard Kalman Filter to nonlinear systems by using linear approximations, making it a practical solution for many real-world applications with nonlinear behavior.

2.1.2 Related Work

In both signal processing and machine learning literature, several approaches have been proposed to merge the predictive capabilities of linear and nonlinear models, especially gradient-boosting trees and ARMA-based models. In [34], the authors combine an ARMA model for initial forecasting with a decision tree model that processes its outputs for classification tasks. This approach is structured in multiple stages. Each stage, feature engineering and model fitting, requires separate optimization, which can be suboptimal when not performed in a joint manner. Moreover, the architecture is designed specifically for classification tasks and does not generalize enough for regression. In [35], the authors address the task of electricity consumption forecasting by using a hybrid approach that combines ARMA and XGBoost models with a two-way switching mechanism. In particular, the method conducts a statistical stationarity test on the input time series and selects the ARMA model if the series is found to be stationary; otherwise, it employs XGBoost for prediction. This way, the models are not only trained disjointly but also lack any mechanism for mutual interaction. As a result, they cannot support each other in tasks such as feature selection, leading to two entirely separate training processes. Furthermore, the methodology is demonstrated solely on electricity consumption data, which limits the evaluation of its effectiveness in other domains and generalization performance of the architecture. There also

exist studies that employ the individual parts of our introduced architecture used as standalone estimators for various prediction tasks. For instance, in [36], the authors employ an ensemble approach that combines ARIMA with GBDT to perform word prediction. However, the ARMA-based model architecture lacks integration with exogenous regressors, which significantly limits its applicability in real-world scenarios where exogenous features play a critical role. Furthermore, unlike our approach, which uses *soft* decision trees (SDTs) as the weak learners in the boosting algorithm, the GBDT used in the aforementioned paper relies on *hard* decision trees. In [37], the authors propose an ensemble model by employing SARIMAX and SVM so that they can achieve accurate rainfall prediction. The use of SVM with a kernel introduces nonlinear modeling capabilities; however, the overall hybrid architecture is sequential, with SARIMAX fitted first and the SVM trained on the resulting residuals. Once again, this approach not only severs the natural link between two models but also results in increased computational complexity due to the lack of joint training and shared optimization.

To conclude, while there have been notable studies in the literature to combine ARMA and decision tree-based models, none of them have introduced a unified framework that enables joint optimization of both components within a single learning architecture. To the best of our knowledge, all existing models rely on disjoint procedures for training and/or feature selection. On the other hand, our primary objective is to eliminate this two-stage process by integrating training and feature selection within a unified state-space model through joint optimization. Moreover, existing models in the literature predominantly rely on *hard* decision trees. However, this introduces limitations in terms of flexibility, particularly in loss function optimization. Due to the non-differentiable nature of hard decision trees, it becomes challenging to employ arbitrary loss functions that may be better suited to specific applications, which is a significant disadvantage compared to differentiable models such as neural networks. By using soft decision process with probabilistic splits in SDTs, all nodes can contribute to the final output and we make entire boosting framework differentiable. Therefore, different from previous studies, our model jointly trains a nonlinear tree-based model (SGBDT) and a linear time-series model (SARIMAX) in an end-to-end

fashion. Both components are unified in a single state-space formulation that supports online learning and is optimized with algorithms such as the EKF. Our experiments demonstrate that the model learns both linear and nonlinear components of the series effectively and achieves substantial performance gains over state-of-the-art machine-learning models on widely used benchmark datasets.

2.2 Material and methods

2.2.1 Problem Statement

We focus on the problem of online, or sequential, regression in the context of time series analysis. In this context, the target values y_t of a time series are observed sequentially, along with corresponding feature vectors $\boldsymbol{x}_t$ that serve as side information, i.e., we assume that $\{\boldsymbol{x}_t\}$ denotes the concatenated form of them. The objective is to learn a predictive function that leverages historical observations y_j and the corresponding feature vectors $\boldsymbol{x}_j$ for all $j \leq t$, in order to predict the next value $\hat{y}_{t+1}$ at each time step. Formally, the prediction function is expressed as:

$$\hat{y}_{t+1} = f\big(\{\ldots, y_{t-1}, y_t\}, \{\ldots, \boldsymbol{x}_{t-1}, \boldsymbol{x}_t\}\big),$$

where f represents the predictive function responsible for estimating $\hat{y}_{t+1}$. At each time step t, we predict $\hat{y}_{t+1}$ by ensembling the output of a linear model (SARIMAX) and a nonlinear model (SGBDT), which enables to exploit the complementary advantages of both modeling approaches.

$$\hat{y}_{t+1} = \hat{y}_{t+1}^{(SARIMAX)} + \hat{y}_{t+1}^{(SGBDT)}, \tag{2.18}$$

where $\hat{y}_{t+1}^{(SARIMAX)}$ and $\hat{y}_{t+1}^{(SGBDT)}$ denote the predictions produced by the SARIMAX and SGBDT models, which capture the linear and nonlinear components of the data, respectively. While ensembling two models, no explicit weighting is

applied to individual model output, as the parameters of both models are optimized jointly. At each time step t, the ensemble model first generates a prediction for $\hat{y}_{t+1}$, after which the loss $\ell(y_{t+1}, \hat{y}_{t+1})$ is computed to quantify the prediction error. The loss function ℓ can be any differentiable error metric, such as the squared error: $\ell(y_{t+1}, \hat{y}_{t+1}) = (y_{t+1} - \hat{y}_{t+1})^2$. The objective is to minimize the total online loss over time, defined as $L_t = \sum_{k=1}^{t} l(y_k, \hat{y}_k)$. For instance, in weather forecasting, y_t represents the temperature of the current day of a city. By using the historical temperatures (commonly referred to as lags), the temperature of the following day is predicted. Moreover, we can use the side information, such as humidity, wind speed, and pressure, to enhance predictive accuracy and reduce the overall loss.

In this study, we propose a novel state-space formulation for the SGBDT. Additionally, we utilize and modify the established state-space representation of the SARIMAX model, which is widely adopted in the time series literature [38]. These two formulations are then integrated into a unified framework, enabling joint optimization of their parameters via the EKF.

For the nonlinear part of our framework, we construct an SGBDT. Gradient boosting is an ensemble learning technique that constructs a strong predictive model by sequentially combining multiple weak learners. Each weak learner is trained by fitting the errors of the preceding weak learner, i.e., it learns to correct the errors of its predecessor. Hence, this iterative process enables to minimize the overall loss function. As illustrated in Fig. 2.1, the algorithm proceeds in a stage-wise manner, where the residuals from the previous iteration serve as the learning target for the next weak learner.

In the case of SGBDT, each weak learner is a Soft Decision Tree (SDT). Notably, in both the signal processing and machine learning literature, hard decision trees are commonly employed as weak learners. Hard decision trees partition the feature space using hard decision boundaries; thus, only a single leaf node determines the final output. Conversely, the SDT, also known as a Neural Tree [39], applies soft decision boundaries by assigning routing probabilities at each node. As a result, all leaf nodes can contribute to the final output, weighted by the

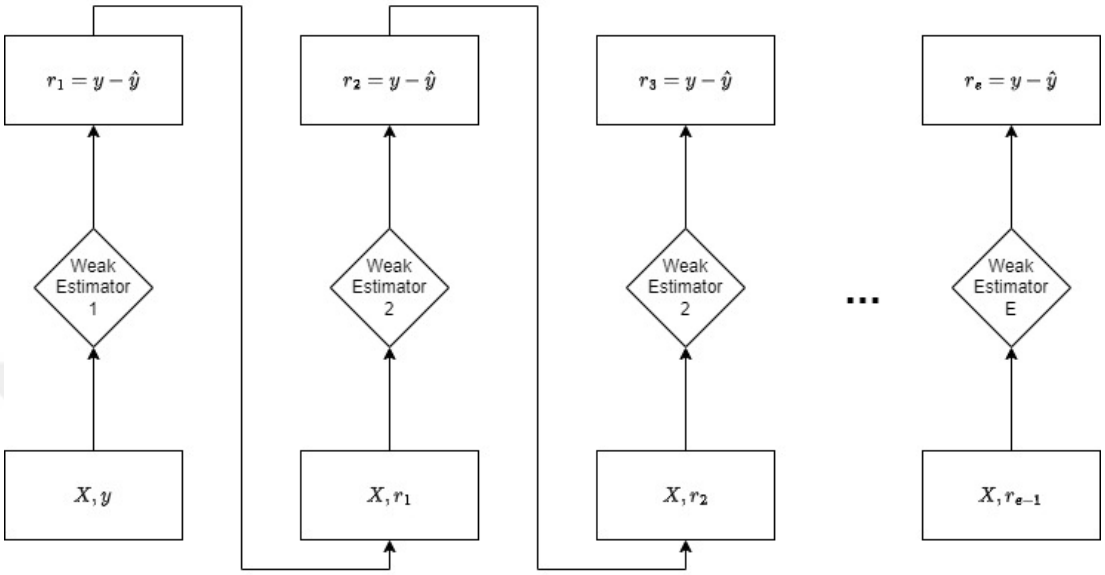

Figure 2.1: Iterative Training Process of the Gradient Boosting Algorithms

probabilities associated with their respective paths.

The SDT consists of a set of internal nodes N and leaf nodes L. At each internal node $n \in N$, the input is processed through a probabilistic decision mechanism to determine the routing probabilities for the child nodes. Specifically, the probability of routing to the left child is modeled as a Bernoulli random variable p_n, defined as

$$p_n = \sigma(\boldsymbol{w}_n^T \boldsymbol{x}_t + b_n) \tag{2.19}$$

where $\boldsymbol{x}_t$ is the input feature vector, $\boldsymbol{w}_n$ and b_n denote the weight vector and bias associated with the internal node n and $\sigma(\cdot)$ represents the sigmoid function, defined as $\sigma(x) = 1/(1 + e^{-x})$. The sigmoid function maps the output to the interval $[0, 1]$ so that we can use the final output for modeling probabilities. Accordingly, the routing probability of the right child is given by $1 - p_n$. For instance, as shown in Fig. 2.2, an SDT of depth 2 comprises three internal nodes and four leaf nodes. At each internal node n, the routing decision is computed using the corresponding weights, biases, and the sigmoid activation function, as

specified in (2.19). To produce rooting probabilities for each node, the input vector is processed through all nodes. The input then reaches each leaf node with an associated path probability, computed as the product of the routing probabilities along the path from the root node to that specific leaf. Each leaf node generates a scalar output, contributing to the final prediction as a weighted sum, where the weights correspond to the respective path probabilities of the leaf nodes, as given by

$$\begin{aligned} f(x) &= \sum_{l=1}^{L} P_l \delta_l \\ &= \sum_{l=1}^{L} \Big(\prod_{n \in N(l)} p_i \Big) \delta_l \\ &= \sum_{l=1}^{L} \Big(\prod_{n \in N(l)} \sigma(\boldsymbol{w}_n^T \boldsymbol{x}_t + b_n) \Big) \delta_l \end{aligned} \tag{2.20}$$

where P_l is the path probability, L denotes the total number of leaves in the tree, $N(l)$ represents the set of internal nodes along the path from the root to the lth leaf node, and δ_l denotes the output of the lth leaf node. Note that δ_l is a learnable parameter. As illustrated in Fig. 2.2, all leaf nodes contribute to the final output.

An SGBDT is constructed by employing multiple SDTs as weak learners. Each SDT is trained sequentially to model the residual errors of its predecessor. Then, the final output is computed as

$$F(\boldsymbol{x}_t) = \sum_{e=1}^{E} \lambda_e f_e(\boldsymbol{x}_t) \tag{2.21}$$

where E is the total number of weak learners, $f_e(\cdot)$ represents the output of the eth weak estimator, λ_e denotes the regularization parameter weighting the output of eth estimator, and $F(\boldsymbol{x}_t)$ is the final output of the SGBDT for the input vector $\boldsymbol{x}_t$. It is important to note that the SGBDT comprises E distinct SDTs, each

characterized by its own set of parameters and individual output. The primary hyperparameters of the SGBDT include the total number of estimators E, the maximum depth permitted for each SDT, and the regularization constant λ.

For the linear component of our framework, we employ the SARIMAX model (Seasonal Auto-Regressive Integrated Moving Average with Exogenous variables) [40], which is a well-established linear approach in time series forecasting. The basic ARMA model, which only includes the Auto-Regressive (AR) and Moving Average (MA) components, is expanded upon by the SARIMAX model. To explain the progressive extensions from the ARMA model to the SARIMAX framework, it is essential to introduce first the ARIMA model as an intermediate development. The ARIMA model advances the ARMA framework by incorporating differencing (I) operations to transform non-stationary time series data into a stationary form. Next, the SARIMAX model further enriches the representation by accounting for seasonal components (S) and integrating exogenous variables (X), thereby offering a more complex modeling capability. The SARIMAX model is defined by two distinct sets of parameters: non-seasonal and seasonal orders. The non-seasonal orders, represented as (p, d, q), correspond to the orders of the auto-regressive (AR) component, the degree of differencing required to achieve stationarity, and the moving average (MA) component, respectively. Similarly, the seasonal orders are denoted as (P, D, Q, s), where P, D, and Q represent the seasonal auto-regressive order, seasonal differencing order, and seasonal moving average order, respectively, and s denotes the length of the seasonal cycle. Furthermore, the model incorporates exogenous variables through the input vector $\boldsymbol{x}_t$, of dimension X. This enables the model to leverage external information while forecasting. The complete SARIMAX model is given by

$$
\begin{aligned}
y_t = c &+ \sum_{i=1}^{p} \alpha_i y_{t-i} + \sum_{i=1}^{q} \theta_i \epsilon_{t-i} \\
&+ \sum_{i=1}^{P} \phi_i y_{t-si} + \sum_{i=1}^{Q} \eta_i \epsilon_{t-si} \\
&+ \sum_{i=1}^{X} \kappa_i \boldsymbol{x}_{t,i} \\
&+ \epsilon_t
\end{aligned}
\tag{2.22}
$$

where c denotes a constant term, ϵ_t represents the white noise process, and the coefficients α_i, θ_i, ϕ, η_i, and κ_i correspond to the parameters of the auto-regressive, moving average, seasonal auto-regressive, seasonal moving average, and exogenous variables, respectively. This notation can be compactly expressed by "Backshift notation", which is a standard convention in time series analysis for denoting lagged terms [41]. The backshift operator B is defined such that $B^i y_t = y_{t-i}$. By using this notation, the SARIMAX model with the same order as in (2.22) can be rewritten as:

$$
\alpha_p(B)\phi_P(B^s)y_t = c + \theta_q(B)\eta_Q(B^s)\epsilon_t + \sum_{i=1}^{X} \kappa_i \boldsymbol{x}_{t,i} \tag{2.23}
$$

where $\alpha_p(B)$, $\phi_P(B^s)$, $\theta_q(B)$, and $\eta_Q(B^s)$ indicate the backshift polynomials corresponding to the non-seasonal auto-regressive, seasonal auto-regressive, non-seasonal moving average, and seasonal moving average components, respectively. These polynomials are explicitly defined as:

$$
\begin{aligned}
\alpha_p(B) &= 1 - \alpha_1 B - \alpha_2 B^2 - \cdots - \alpha_p B^p \\
\phi_P(B^s) &= 1 - \phi_1 B^s - \phi_2 B^{2s} - \cdots - \phi_P B^{Ps} \\
\theta_q(B) &= 1 - \theta_1 B - \theta_2 B^2 - \cdots - \theta_q B^q \\
\eta_Q(B^s) &= 1 - \eta_1 B^s - \eta_2 B^{2s} - \cdots - \eta_Q B^{Qs}
\end{aligned}
\tag{2.24}
$$

Additionally, when differencing is applied to achieve stationarity, the differencing

operator Δ_d is defined as $(1-B)^d$. Accordingly, the differenced time series is given by $\Delta_d(B)\Delta_D(B^s)y_t = (1-B)^d(1-B^s)^D y_t$, where d and D denote the orders of non-seasonal and seasonal differencing, respectively.

In the subsequent section, we introduce a novel state-space formulation for both the SGBDT and SARIMAX models. These representations are then unified within a single framework to facilitate joint optimization. Building upon this unified structure, we present an estimation method for online prediction based on the EKF algorithm.

2.2.2 The proposed model

Combining SGBDT and SARIMAX models into a unified ensemble is inherently challenging because of their fundamentally different optimization strategies. The SGBDT consists of SDTs as weak learners. Different from traditional hard decision trees, SDTs employ a hierarchical structure with probabilistic branching at each node, enabling them to generate probabilistic outputs. In other words, it establishes soft decision boundaries rather than hard ones. SDTs are typically optimized using gradient-based methods, such as stochastic gradient descent. On the other hand, SARIMAX, which is an enhanced form of the ARIMA model incorporating seasonal components and exogenous variables, generally employs Bayesian estimation techniques alongside iterative optimization methods such as Maximum Likelihood Estimation (MLE). In addition, gradient-based optimization algorithms such as BFGS and its limited-memory variant L-BFGS [42] can be employed in order to optimize the parameters of the SARIMAX. These methods, in principle, enable the joint optimization of SARIMAX and SGBDT components, thus providing a potential pathway for effective ensembling. However, in practice, these gradient-based methods are not preferred because of the complexity of handling moving average (MA) components within the gradient calculation. MA terms are nonlinear functions of past prediction errors. Also, in higher-order models, their gradients can become discontinuous and computationally unstable. Gradient calculation becomes even more difficult with the addition of seasonal

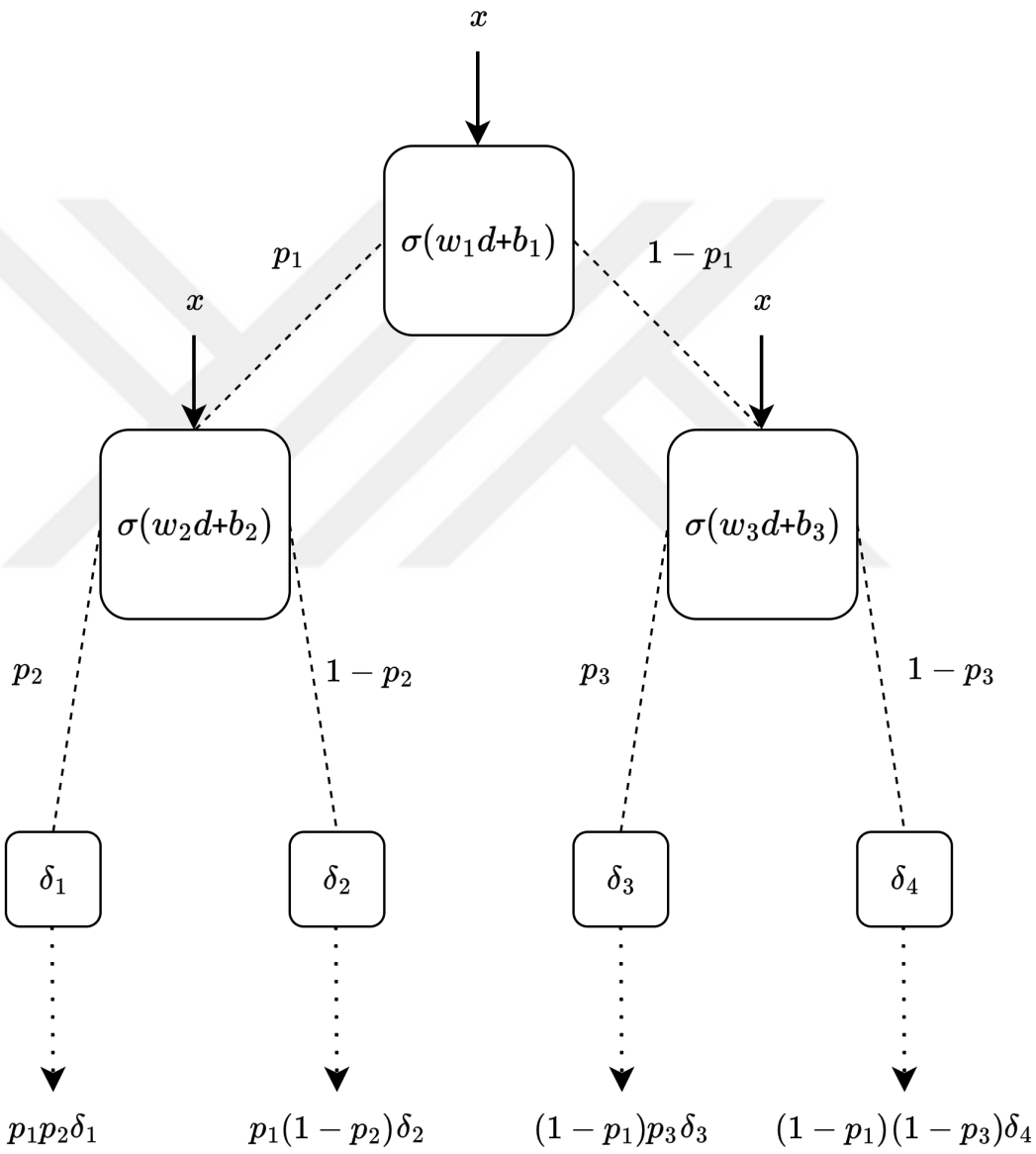

Figure 2.2: The forward pass of an SDT with a depth of 2. The input vector $\boldsymbol{x}$ traverses all internal nodes, generating the corresponding routing probabilities, denoted as p_n. All the leaf nodes contribute to the final output in proportion to their path probabilities calculated during traversal.

components. These components introduce long-range dependencies that pose significant challenges for gradient-based optimization methods, which often struggle with the vanishing gradient problem, especially with large seasonal periods.

To overcome these limitations, Kalman filter-based techniques offer an efficient optimization alternative while ensembling SGBDT and SARIMAX in a unified framework. These methods are known as rapid convergence and robust performance [43], particularly when data is limited [44]. The standard Kalman filter operates by estimating the state and associated uncertainty of a system and updating these estimates in an online manner. However, it has a linear formulation, which makes it unsuitable to optimize nonlinear models such as SGBDT. To address this, we choose the Extended Kalman Filter (EKF), which supports nonlinear optimization [45], and we develop novel and unified state-space representations for both the SARIMAX and SGBDT components, enabling their integration within the EKF framework.

2.2.2.1 State-Space Representations

In this section, we introduce the state-space representations of both the SGBDT and the SARIMAX model. These representations are then systematically integrated into a unified modeling framework.

2.2.2.1.1 SARIMAX

The SARIMAX model extends the classical ARMA framework by incorporating both seasonal components and exogenous variables. Accordingly, we progress step by step from the foundational state-space representations of the ARMA model to the more comprehensive SARIMAX formulation. While the ARMA model is defined solely by its auto-regressive (AR) and moving average (MA) components of orders (p, q), the SARIMAX model includes both non-seasonal and seasonal dynamics. These orders of both dynamics are represented by (p, d, q) for the non-seasonal part and (P, D, Q, s) for the seasonal part, where p and q denote the orders of the non-seasonal AR and MA terms, P and Q correspond to the

seasonal AR and MA terms, d and D represent the non-seasonal and seasonal differencing orders, and s specifies the length of the seasonal period.

In the standard state-space representation of the ARMA model, parameters including α_i and θ_i for the AR and MA components are considered fixed (i.e., time-invariant) coefficients, as demonstrated in (2.22). However, in our proposed approach, these parameters are no longer treated as constant. To support an online learning setting, where the model must continuously adapt to newly arriving data, it is necessary to introduce time-varying parameters. Specifically, the static coefficients α_i and θ_i are replaced with their dynamic counterparts $\alpha_{t,i}$ and $\theta_{t,i}$, which evolve over time. Based on this modification, the ARMA equation can be reformulated as

$$y_t = c + \sum_{i=1}^{p} \alpha_{t,i} y_{t-i} + \sum_{i=1}^{q} \theta_{t,i} \epsilon_{t-i} + \epsilon_t, \tag{2.25}$$

where $\alpha_{t,i}$ and $\theta_{t,i}$ denote the time-varying AR and MA coefficients, respectively. This formulation allows the model to adapt dynamically over time. This concept is further extended to the SARIMAX formulation by replacing all static parameters in Equation (2.22) with their corresponding time-varying counterparts, given as:

$$\begin{aligned} y_t = c &+ \sum_{i=1}^{p} \alpha_{t,i} y_{t-i} + \sum_{i=1}^{q} \theta_{t,i} \epsilon_{t-i} \\ &+ \sum_{i=1}^{P} \phi_{t,i} y_{t-si} + \sum_{i=1}^{Q} \eta_{t,i} \epsilon_{t-si} \\ &+ \sum_{i=1}^{X} \kappa_{t,i} \boldsymbol{x}_{t,i} \\ &+ \epsilon_t \end{aligned} \tag{2.26}$$

where $\alpha_{t,i}$, $\theta_{t,i}$, $\phi_{t,i}$ and $\eta_{t,i}$ denote the time varying coefficients for AR, MA, seasonal AR, seasonal MA parameters, respectively, while $\kappa_{t,i}$ represents the time

varying parameters associated with the exogenous features. To express in a state-space form, the state vector is defined as:

$$\boldsymbol{z}_t^{\text{SARIMAX}} := [\boldsymbol{\alpha}_t^T, \boldsymbol{\theta}_t^T, \boldsymbol{\phi}_t^T, \boldsymbol{\eta}_t^T, \boldsymbol{\kappa}_t^T]^T \in \mathbb{R}^{p+q+P+Q+X}, \tag{2.27}$$

where $\boldsymbol{\alpha}_t$, $\boldsymbol{\theta}_t$, $\boldsymbol{\phi}_t$, $\boldsymbol{\eta}_t$ and $\boldsymbol{\kappa}_t$ denote the vectorized forms of the corresponding time varying parameters. The total dimension of the state vector is given by $V_{SARIMAX} = p + q + P + Q + X$, which corresponds exactly to the number of parameters that must be estimated in the SARIMAX model. Particularly, the individual parameter vectors are defined as follows:

$$\begin{aligned}
\boldsymbol{\alpha}_t &= [\alpha_{t,1}, \alpha_{t,2}, \ldots, \alpha_{t,p}]^T, \\
\boldsymbol{\theta}_t &= [\theta_{t,1}, \theta_{t,2}, \ldots, \theta_{t,q}]^T, \\
\boldsymbol{\phi}_t &= [\phi_{t,1}, \phi_{t,2}, \ldots, \phi_{t,P}]^T, \\
\boldsymbol{\eta}_t &= [\eta_{t,1}, \eta_{t,2}, \ldots, \eta_{t,Q}]^T, \\
\boldsymbol{\kappa}_t &= [\kappa_{t,1}, \kappa_{t,2}, \ldots, \kappa_{t,X}]^T.
\end{aligned} \tag{2.28}$$

Based on the state vector $\boldsymbol{z}_t^{\text{SARIMAX}}$, we introduce the SARIMAX model in a state-space form, consisting of the state transition and the measurement equations as:

$$\boldsymbol{z}_t^{\text{SARIMAX}} = \boldsymbol{z}_{t-1}^{\text{SARIMAX}} + \boldsymbol{e}_t \tag{2.29}$$

$$\hat{y}_t^{\text{SARIMAX}} = \boldsymbol{h}_t^T \boldsymbol{z}_t^{\text{SARIMAX}} + \epsilon_t \tag{2.30}$$

where e_t is an independently and identically distributed (i.i.d) noise vector, ϵ_t denotes a white noise process, and $\boldsymbol{h}_t^T$ a time-dependent feature vector composed of past observations, past errors, and exogenous variables, defined as:

$$\begin{aligned}
\boldsymbol{h}_t^T = [&y_{t-1} \ldots y_{t-p},\ e_{t-1} \ldots e_{t-q}, \\
&y_{t-s} \ldots y_{t-sP},\ e_{t-s} \ldots e_{t-sQ}, \\
&x_{t,1} \ldots x_{t,X}]^T.
\end{aligned}$$

2.2.2.1.2 Soft Gradient Boosting Decision Tree

SGBDT represents a variant of the traditional gradient boosting decision tree algorithm, distinguished by its use of SDTs instead of traditional decision trees. Different from a traditional decision tree, an SDT assigns a probability to each routing decision at internal nodes, enabling information to propagate to all leaf nodes and thereby incorporate their weighted contributions into the final prediction. An SDT comprises N internal nodes and L leaf nodes. Each internal node is parameterized by a weight vector $\boldsymbol{w}$ and a bias b, which define the probabilistic routing function given by (2.19). Correspondingly, each leaf node holds a learnable parameter δ, as shown in (2.20). Therefore, the number of these learnable parameters grows with the depth of the tree. For instance, a depth-d SDT contains $2^d - 1$ internal nodes and 2^d leaf nodes, leading to $2^d - 1$ weights and biases, and 2^d leaf nodes' specific parameters. Notably, in our formulation, all these parameters are modeled as time-varying, which enables us to integrate them into a dynamic state-space model. We organize time varying weights $w_{t,i}$, biases $b_{t,i}$ and leaf specific parameters $\delta_{t,i}$ of the SDT into parameter vectors as

$$\begin{aligned} \boldsymbol{\omega}_t &= [w_{t,1}, w_{t,2} \dots w_{t,2^d-1}]^T, \\ \boldsymbol{\beta}_t &= [b_{t,1}, b_{t,2} \dots b_{t,2^d-1}]^T, \\ \boldsymbol{\delta}_t &= [\delta_{t,1}, \delta_{t,2} \dots \delta_{t,2^d}]^T \end{aligned} \tag{2.31}$$

where $\boldsymbol{\omega}_t$, $\boldsymbol{\beta}_t$, and $\boldsymbol{\delta}_t$ denote the time-varying vectors of weights, biases, and leaf node parameters, respectively. The vector $\boldsymbol{\omega}_t$ has a dimensionality of $X \cdot (2^d - 1)$, where X is the size of the weight vector associated with each internal node, and $2^d - 1$ is the total number of internal nodes. The bias vector $\boldsymbol{\beta}_t$ contains $2^d - 1$ elements, reflecting the number of internal nodes, while the leaf parameter vector $\boldsymbol{\delta}_t$ consists of 2^d elements, reflecting the number of leaf nodes. Thus, the total number of parameters in an SDT is given by $V_{SDT} = X \cdot (2^d - 1) + 2 \cdot 2^d - 1$. These vectors are then concatenated to construct the overall state vector of the SDT as

$$\boldsymbol{z}_t^{\text{SDT}} := [\boldsymbol{\omega}_t^T, \boldsymbol{\beta}_t^T, \boldsymbol{\delta}_t^T]^T \in \mathbb{R}^{V_{SDT}}, \tag{2.32}$$

which enables us to express the state transition and measurement equations as

$$\boldsymbol{z}_t^{\mathrm{SDT}} = \boldsymbol{z}_{t-1}^{\mathrm{SDT}} + \hat{\boldsymbol{e}}_t \tag{2.33}$$

$$\hat{y}_t^{\mathrm{SDT}} = \zeta(\boldsymbol{x}_t,\ \boldsymbol{z}_t^{\mathrm{SDT}}) + \hat{\epsilon}_t \tag{2.34}$$

where $\hat{\boldsymbol{e}}_t$ denotes the i.i.d noise vector, $\hat{\epsilon}_t$ represents the white noise process, and ζ is the nonlinear function that defines the nonlinear relationship in (2.20) using the exogenous input features $\boldsymbol{x}_t$ along with the parameter vector.

To extend the formulation from a single SDT to an SGBDT, it is essential to generalize the derivation further. In the gradient boosting algorithm, let E denote the total number of weak learners, where each learner corresponds to an individual SDT. Since each SDT maintains its own distinct set of weights, biases, and leaf-specific learnable parameters, it is essential to expand our state vector accordingly to account for all E trees as

$$\begin{aligned}\boldsymbol{z}_t^{\mathrm{SGBDT}} := [\boldsymbol{\omega}_{1,t}^T, \boldsymbol{\beta}_{1,t}^T, \boldsymbol{\delta}_{1,t}^T, \ldots, \quad \boldsymbol{\omega}_{E,t}^T, \boldsymbol{\beta}_{E,t}^T, \boldsymbol{\delta}_{E,t}^T]^T \\ \in \mathbb{R}^{E \cdot V_{SDT}}\end{aligned} \tag{2.35}$$

where the total dimensionality of the SGBDT state vector is multiplied by a factor of E relative to that of a single SDT. Upon integrating the gradient boosting algorithm, the state transition model remains applicable with minor modifications, given as

$$\boldsymbol{z}_t^{\mathrm{SGBDT}} = \boldsymbol{z}_{t-1}^{\mathrm{SGBDT}} + \hat{\boldsymbol{e}}_t. \tag{2.36}$$

However, this results in substantial modifications in the measurement equation based on (2.21) as

$$\hat{y}_t^{\mathrm{SGBDT}} = \sum_{e=1}^{E} \lambda_e \zeta_e \left(\boldsymbol{x}_t,\ \boldsymbol{z}^{\mathrm{SGBDT}}_{\left\{(e-1)\cdot V_{SDT}:e\cdot V_{SDT}\right\},t} \right) + \hat{\epsilon}_t \tag{2.37}$$

where E represents the total number of weak estimators, $\zeta_e(\boldsymbol{x}_t, \cdot)$ refers to the nonlinear function associated with the eth estimator, as previously defined in (2.34), that operates on the input feature vector $\boldsymbol{x}_t$ along with a specific segment of the state vector $\boldsymbol{z}_t^{\text{SGBDT}}$, and λ_e denotes the regularization constant that determines the individual contribution of each weak estimator to the overall model output.

In order to enable joint optimization of the SARIMAX and SGBDT models, we concatenate their respective state vectors into a single, unified parameter vector of size $V = V_{SARIMAX} + E \cdot V_{SDT}$

$$\boldsymbol{z}_t := [(\boldsymbol{z}_t^{\text{SARIMAX}})^T, (\boldsymbol{z}_t^{\text{SGBDT}})^T]^T \in \mathbb{R}^V \tag{2.38}$$

where the expanded form of the joint state vector is given by:

$$\begin{aligned} \boldsymbol{z}_t := [&\boldsymbol{\alpha}_t^T, \boldsymbol{\theta}_t^T, \boldsymbol{\phi}_t^T, \boldsymbol{\eta}_t^T, \boldsymbol{\kappa}_t^T, \\ &\boldsymbol{\omega}_{1,t}^T, \boldsymbol{\beta}_{1,t}^T, \boldsymbol{\delta}_{1,t}^T \\ &\dots \\ &\boldsymbol{\omega}_{E,t}^T, \boldsymbol{\beta}_{E,t}^T, \boldsymbol{\delta}_{E,t}^T] \in \mathbb{R}^V, \end{aligned} \tag{2.39}$$

where the vectors $\boldsymbol{\alpha}_t$, $\boldsymbol{\theta}_t$, $\boldsymbol{\phi}_t$, $\boldsymbol{\eta}_t$, and $\boldsymbol{\kappa}_t$ represent the parameters associated with the SARIMAX, and $\boldsymbol{\omega}_{c,t}$, $\boldsymbol{\beta}_{c,t}$, and $\boldsymbol{\delta}_{c,t}$ denote the weights, biases, and leaf parameters of cth weak learner in the SGBDT. Furthermore, the resulting joint state-space representation that comprises both the state transition and measurement equations becomes

$$\begin{aligned} \boldsymbol{z}_t &= \boldsymbol{z}_{t-1} + \hat{e}_t \\ &= \varrho(\boldsymbol{z}_{t-1}) + \hat{e}_t \end{aligned} \tag{2.40}$$

$$\begin{aligned} \hat{y}_t &= \hat{y}_t^{SARIMAX} + \hat{y}_t^{SGBDT} + \hat{\epsilon}_t \\ &= \boldsymbol{h}_t^T \boldsymbol{z}_t^{\text{SARIMAX}} + \zeta(\boldsymbol{x}_t,\ \boldsymbol{z}_t^{\text{SGBDT}}) + \hat{\epsilon}_t \\ &= \rho(\boldsymbol{h}_t, \boldsymbol{x}_t, \boldsymbol{z}_t) + \hat{\epsilon}_t \end{aligned} \tag{2.41}$$

where ϱ denotes the state transition function, responsible for modeling the temporal evolution of the state vector $\boldsymbol{z}_t$, and is commonly referred to as the process model, while ρ represents the measurement model, which integrates the measurement equation of both the SARIMAX and SGBDT. Specifically, ρ performs a linear inner product between the SARIMAX component $\boldsymbol{z}_t^{\text{SARIMAX}}$ and the design vector $\boldsymbol{h}_t$, while applying a nonlinear transformation $\zeta(\cdot)$ to the exogenous input $\boldsymbol{x}_t$ by using the weight $\boldsymbol{\omega}_t$ and bias $\boldsymbol{\beta}_t$ parameters contained in $\boldsymbol{z}_t^{\text{SGBDT}}$.

In the following section, we introduce the construction of the jointly optimized model, which leverages the previously derived state-space representations and employs the Extended Kalman Filter (EKF) for sequential parameter estimation.

2.2.2.2 Estimation with Extended Kalman Filter

In this section, we present the EKF algorithm to estimate the state vector $\boldsymbol{z}_t$ in an online manner using the observed measurements $\hat{y}_t$. While the traditional Kalman Filter is suited for systems with linear dynamics, where both state transition and measurement equations are linear, the EKF extends the Kalman Filter to accommodate nonlinear systems [46]. This is achieved by linearizing the nonlinear functions at each time step through a first-order Taylor series expansion. To implement the EKF algorithm, the state-space formulation of the unified model is required, as derived in Section 2.2.2. In the context of the SARIMAX–SGBDT ensemble model, equations (2.40) and (2.41) present the final forms of the state transition and measurement equations for the state vector $\boldsymbol{z}_t$ and corresponding measurement $\hat{y}_t$, respectively. To proceed with the EKF updates, we linearize both the process model ϱ and the measurement model ρ at each time step via first-order Taylor series expansion. This procedure yields the Jacobian matrices associated with the state transition and measurement functions, which are computed as

$$\begin{aligned} \boldsymbol{G}_t &= \frac{\partial \varrho}{\partial \boldsymbol{z}} \\ [\boldsymbol{G}_t]_{i,j} &= \frac{\partial \varrho_i}{\partial \boldsymbol{z}_j} \in \mathbb{R}^{V \times V}, \end{aligned} \tag{2.42}$$

$$\begin{aligned} \boldsymbol{P}_t &= \frac{\partial \rho}{\partial \boldsymbol{z}} \\ [\boldsymbol{P}_t]_{i,j} &= \frac{\partial \rho_i}{\partial \boldsymbol{z}_j} \in \mathbb{R}^{1 \times V} \end{aligned} \tag{2.43}$$

where V denotes the size of the state vector as defined in equation(2.38), $[\boldsymbol{G}_t]_{i,j}$ corresponds to the partial derivative of the ith component of the process model ϱ with respect to the jth component of the state vector $\boldsymbol{z}$, and similarly, $[\boldsymbol{P}_t]_{i,j}$ denotes the partial derivative of the ith component of the measurement model ρ with respect to $\boldsymbol{z}_j$. These Jacobian matrices are essential for propagating uncertainty and updating state estimates within the EKF.

Here, the Extended Kalman Filter (EKF) algorithm can be initiated to generate sequential state estimations. The EKF operates through two fundamental steps: the Prediction Step and the Update Step. During the Prediction Step, the goal is to estimate the current state based on the previous state as defined in (2.40). Then, the state covariance matrix is predicted to quantify the associated uncertainty of the state estimation. The predicted covariance $\boldsymbol{Z}_{t|t-1}$ is computed from the previous state covariance matrix $\boldsymbol{Z}_{t-1|t-1}$, using the Jacobian of the process model $\boldsymbol{G}_t$, given as

$$\boldsymbol{Z}_{t|t-1} = \boldsymbol{G}_t \boldsymbol{Z}_{t-1|t-1} \boldsymbol{G}_t^T + \boldsymbol{C}_t \in \mathbb{R}^{V \times V}, \tag{2.44}$$

where $\boldsymbol{C}_t$ denotes the noise covariance matrix accounting for model uncertainty of the state estimation.

Next, the Update Step consists of 5 key parts. The first step involves calculating the measurement residual, which quantifies the difference between the predicted measurement obtained from (2.41) and the actual observation. This

residual indicates the degree of deviation between the predicted and observed values. It is computed as

$$\begin{aligned} r_t &= y_t - \hat{y}_t \\ &= y_t - \rho(\boldsymbol{h}_t, \boldsymbol{x}_t, \boldsymbol{z}_t + \hat{\epsilon}_t) \end{aligned} \tag{2.45}$$

where y_t denotes the actual observation at time t, and $\hat{y}_t$ represents the predicted measurement. The measurement residual r_t is a scalar that reflects the prediction error at time t. For consistency with matrix-based computations in the EKF, this scalar is expressed as a 1×1 vector, denoted $\boldsymbol{r}_t \in \mathbb{R}^{1\times 1}$. Next, to characterize the uncertainty associated with this prediction error, we compute the residual covariance derived from the current state covariance $\boldsymbol{Z}_{t|t-1}$ and Jacobian matrix of the measurement model $\boldsymbol{P}_t$, given as

$$\boldsymbol{R}_t = \boldsymbol{P}_t \boldsymbol{Z}_{t|t-1} \boldsymbol{P}_t^T + \boldsymbol{Q}_t \in \mathbb{R}^{1\times 1}, \tag{2.46}$$

where $\boldsymbol{Q}_t$ denotes the noise covariance matrix of the measurement. As a result of the preceding steps, we obtained the deviation of the prediction and the covariance of the residual. The next step is to compute the Kalman Gain, which determines the optimal adjustment to the predicted state. The Kalman Gain accounts for the uncertainties in both the state estimation and the measurement by incorporating the state covariance $\boldsymbol{Z}_{t|t-1}$ and residual covariance $\boldsymbol{R}_t$, and is given as

$$\boldsymbol{K}_t = \boldsymbol{Z}_{t|t-1} \boldsymbol{P}_t^T \boldsymbol{R}_t^{-1} \in \mathbb{R}^{V\times 1}. \tag{2.47}$$

With the Kalman Gain $\boldsymbol{K}_t$ computed, the final two steps of the EKF involve updating the state estimation and the associated covariance. These steps are called the state update and covariance update steps, and are given by:

$$\boldsymbol{z}_{t|t} = \boldsymbol{z}_t + \boldsymbol{K}_t \boldsymbol{r}_t \in \mathbb{R}^{V\times 1}. \tag{2.48}$$

$$\boldsymbol{Z}_{t|t} = (\boldsymbol{I} - \boldsymbol{K}_t \boldsymbol{P}_t)\boldsymbol{Z}_{t|t-1} \in \mathbb{R}^{V \times V}, \tag{2.49}$$

where $\boldsymbol{I}$ represents the identity matrix of dimension $V \times V$. These updates refine the predicted state and adjust the associated covariance, resulting in a more accurate and informed estimate of both the state and its uncertainty.

Hence, the complete EKF algorithm for online state estimation is summarized in Algorithm 1, which details the step-by-step derivations required to estimate both the state vector and its associated covariance matrix. At each time step t, the algorithm starts with the prediction of the state vector and computes the measurement obtained by the predicted state. This is followed by the calculation of the Jacobian matrices for both the process and measurement models, which are required to linearize the system dynamics. Using the Jacobian matrix belonging to the process model, the algorithm then predicts the state covariance matrix. Next, in the update phase, the measurement residual and its covariance are calculated. These are used to derive the Kalman Gain, which enables us to update the state estimate and the state covariance matrix in an optimal way. This recursive process enables continuous refinement of state estimates as new data becomes available. Progressively, we obtain more accurate predictions over time.

Algorithm 1 Online Prediction of EKF

Require: T Number of Measurements

Initialize State estimate $\boldsymbol{z}_0$,

Initialize covariance estimate $\boldsymbol{Z}_0$,

Initialize Noise covariance for state $\boldsymbol{C}_0$,

Initialize Noise covariance for Measurement $\boldsymbol{Q}_0$

Define Process model $\varrho(\cdot)$,

Define Measurement model $\rho(\cdot)$,

Ensure: Estimated state $\boldsymbol{z}_t$ and covariance $\boldsymbol{Z}_t$ at each time step t

for $t = 1$ to T **do**

▷ **Prediction Steps:**

$\boldsymbol{z}_t = \varrho(\boldsymbol{z}_{t-1}) + \hat{e}_t$ ▷ Predict the next state

$\hat{y}_t = \rho(\boldsymbol{h}_t, \boldsymbol{x}_t, \boldsymbol{z}_t) + \hat{\epsilon}_t$ ▷ Compute the measurement

$\boldsymbol{G}_t = \frac{\partial \varrho}{\partial \boldsymbol{z}}$ ▷ Compute Jacobian of the process model

$\boldsymbol{P}_t = \frac{\partial \rho}{\partial \boldsymbol{z}}$ ▷ Compute Jacobian of the measurement model

$\boldsymbol{Z}_{t|t-1} = \boldsymbol{G}_t \boldsymbol{Z}_{t-1|t-1} \boldsymbol{G}_t^T + \boldsymbol{C}_t$ ▷ Predict the state covariance

▷ **Update Steps:**

$r_t = y_t - \hat{y}_t$ ▷ Compute the measurement residual

$\boldsymbol{r}_t = \begin{bmatrix} r_t \end{bmatrix}$ ▷ Convert to 1×1 vector

$\boldsymbol{R}_t = \boldsymbol{P}_t \boldsymbol{Z}_{t|t-1} \boldsymbol{P}_t^T + \boldsymbol{Q}_t$ ▷ Compute the residual covariance

$\boldsymbol{K}_t = \boldsymbol{Z}_{t|t-1} \boldsymbol{P}_t^T \boldsymbol{R}_t^{-1}$ ▷ Compute the Kalman gain

$\boldsymbol{z}_{t|t} = \boldsymbol{z}_t + \boldsymbol{K}_t \boldsymbol{r}_t$ ▷ Update the state estimate

$\boldsymbol{Z}_{t|t} = (\boldsymbol{I} - \boldsymbol{K}_t \boldsymbol{P}_t) \boldsymbol{Z}_{t|t-1}$ ▷ Update the state covariance

end for

Chapter 3

Experiments

We conduct a comprehensive experimental evaluation to assess the performance and generalization of the proposed model, referred to as SGBDT + SARIMAX, under different scenarios. We organize the evaluation framework into 4 distinct parts. In the first part, we investigate the learning behavior and forecasting capability of our ensemble model on real-world datasets. Specifically, we choose three publicly available datasets: the Turkey Gas Demand Dataset, the Elevators Dataset, and the Pumadyn Dataset. To benchmark our model, we compare its performance against a diverse set of baseline and state-of-the-art (SOTA) models. As baseline models, we include a naive forecaster, a linear regression model, and a traditional (hard) decision tree. For comparison with the SOTA models, we evaluate against advanced gradient boosting frameworks such as LightGBM [16], XGBoost [21], and SARIMAX [40] as an advanced statistical model. In addition, we incorporate prominent deep learning-based time series forecasting methods such as Artificial Neural Network (ANN), DeepAR [28], Informer [12], and iTransformer [13]. These models have been commonly preferred in recent years for their capacity to capture complex temporal dependencies and nonlinearity [47]. In particular, Informer and iTransformer leverage transformer-based architectures that are well suited for modeling long-range temporal relationships. Thus, we could comprehensively evaluate the practical effectiveness and robustness of our model in diverse and dynamic forecasting environments. In the second part,

we use the well-established M4 Competition Dataset [48], which offers a diverse collection of time series data that span multiple temporal resolutions, including yearly, weekly, daily, and hourly. The M4 Competition Dataset allows us to systematically analyze our model's performance under diverse forecasting challenges and across different frequencies. Next, in the third part, we perform an ablation study to isolate and understand the contributions of individual components of the proposed architecture. This includes separate evaluations of the SDT, its gradient-boosted variant (SGBDT), and the SARIMAX model. Furthermore, we compare the performance of the jointly optimized SGBDT + SARIMAX model with a version trained in a disjoint manner, in order to experimentally evaluate the benefits of joint optimization on overall model performance. Finally, we present a real-world 5G application using the 5G YouTube-Live dataset. We begin with an Exploratory Data Analysis (EDA) to gain information about the characteristics of the dataset and guide our feature extraction process. We expand our model comparison by including additional forecasting models, namely SVR, RNN, LSTM, and GRU.

3.1 Real-Life Datasets

In this section, we evaluate the proposed model using three different real-world datasets belonging to different application domains. The evaluation is conducted in an online learning setting, in which the model is continually updated as new data becomes available. This enables it to respond and adapt effectively to evolving patterns over time.

- **Elevators Dataset [49]**
  The Elevators dataset is collected from a mission involving the control of an F-16 aircraft. The objective is to predict an action taken on the elevators of the aircraft. With 18 input features, the dataset consists of 16,599 instances in order to predict the target value.

- **Natural Gas Demand of Turkey**

  The Gas Demand dataset includes the total residential natural gas consumption in Turkey from 2018 to 2020. It provides meteorological features such as wind speed and temperature. In addition, the feature set includes calendar-based features such as day, month, and weekend indicators. Moreover, we extract various target-related features, such as lags, rolling means, and rolling standard deviations, based on historical consumption.

- **Pumadyn Dataset [49]**

  The Pumadyn dataset is based on the physical dynamics of the robotic arm Puma 560. The prediction task involves estimating the angular acceleration of the arm's links. The dataset offers two configurations, containing 8 and 32 features, respectively. In our experiments, we use the 32-feature version, which includes angular positions, velocities, and torques of the robot arm.

We design a rigorous and comprehensive experimental setup so that we can compare a wide range of models and establish a reliable performance benchmark. In pursuit of this goal, we consider ten models: the Naive model, Linear Regressor, Decision Trees, Artificial Neural Networks (ANN), DeepAR, Informer, iTransformer, SARIMAX, LightGBM, and XGBoost. The simplest and fundamental model is the Naive model, which assigns the forecast $\hat{y}_{t+1} = y_t$ at every time step. That is, it uses the most recent observation as the prediction for the next value. The linear regressor implemented using ordinary least squares (OLS) provides a straightforward yet informative baseline for comparison [50]. Also, the SARIMAX is a linear model specifically designed for time series forecasting, incorporating seasonal components (S), auto-regressive terms (AR), moving average terms (MA), and differencing (I) to induce stationarity, along with exogenous variables (X) [41]. We optimize both nonseasonal and seasonal parameters via grid search over a validation set. Next, to represent the class of nonlinear and differentiable models, we employ Artificial Neural Networks (ANNs), which are particularly effective in capturing complex, nonlinear patterns in data [51]. For ANN tuning, we adjust the learning rate and the number of hidden layers while using the ReLU activation function [24] and the Adam optimizer [52]. As advanced deep learning models, we include DeepAR, Informer, and iTransformer.

DeepAr is based on a recurrent neural network architecture and predicts a probabilistic distribution by using the auto-regressive input. Informer and iTransformer both have the transformer architecture and perform highly effectively at modeling long-range dependencies in time series data. For these advanced deep learning models, we perform hyperparameter optimization over learning rates and architecture-specific layer configurations. We also include a comparison with tree-based models, ranging from traditional decision trees to advanced gradient boosting methods. Traditional decision trees partition the feature space through recursive splits. We tune the max depth and the minimum number of samples required to split a node. Then, we choose two leading gradient boosting frameworks, namely LightGBM and XGBoost. Both are decision tree-based ensembles optimized via boosting algorithms. While LightGBM employs a leaf-wise growth strategy, XGBoost adopts a level-wise growth pattern. We search for the hyperparameters, e.g., the number of estimators, maximum tree depth, number of leaves, and the regularization term. Finally, for hyperparameter optimization of our proposed SGBDT + SARIMAX model, we perform a joint search over the same SARIMAX orders, e.g., nonseasonal and seasonal orders, and SGBDT-specific parameters, including the number of estimators and maximum depth of the SDTs used in the ensemble.

To ensure a fair experimental setup, we divide each dataset into training and test subsets. We allocate the last 30% of the dataset as the test set. Subsequently, 30% of the training data is reserved for validation, while the remaining 70% is used to train the model. Before the final evaluation on the test set, each model is retrained using all the training data for optimal performance. We evaluate model performance using the Mean Squared Error (MSE) metric on the test set, which is a widely used error metric for regression tasks. The MSE is formally defined as

$$\mathrm{MSE}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 , \tag{3.1}$$

where y_i and $\hat{y}_i$ represent the target and predicted values, respectively, and n denotes the total number of test samples. In Table 3.1, we demonstrate the MSE values obtained on the held-out test set. Our proposed model consistently achieves the lowest error across all three datasets. For example, LightGBM and

XGBoost demonstrate similar MSE performance as 4.60×10^{-3} and 4.74×10^{-3}, respectively, while our model shows an approximate 37.61% and 39.45% reduction in error relative to these SOTA models. This significant improvement is attributed to the joint optimization of SGBDT and SARIMAX in a unified framework. In the Turkey Gas Demand and Pumadyn datasets, while the absolute margin of improvement is smaller, our model still outperforms SOTA baselines. Particularly, in the Pumadyn dataset, it achieves a 91.54% reduction in error when compared to the baseline linear regression model. On the other hand, models designed to capture temporal dependencies, such as SARIMAX, DeepAR, Informer, and iTransformer, underperform on this dataset. This is primarily due to the relatively weak temporal dependencies in the dataset, which instead exhibit strong feature-based dependencies. Furthermore, the Naive model's poor performance highlights the limited dependency on past observations, which reinforces the validity of our results. To further evaluate predictive performance over time, we also compute the cumulative mean squared error up to time T, given as

$$CumMSE(\boldsymbol{y_T}, \hat{\boldsymbol{y}}_{\boldsymbol{T}}) = \frac{\sum_{i=1}^{T}(y_i - \hat{y}_i)^2}{T} \tag{3.2}$$

where T denotes the final time index for the cumulative error calculation, $\boldsymbol{y_T}$ and $\hat{\boldsymbol{y}}_{\boldsymbol{T}}$ represent the target and predicted value vectors of length T, respectively. Dividing by T normalizes the total squared error, allowing for a smoother and more interpretable evaluation of prediction performance over time. We demonstrate CumMSE at each time step of the test set for the Gas Demand Dataset in Fig. 3.1. We observe that our proposed model consistently maintains the lowest CumMSE across all time steps in the test set. Notably, it demonstrates more stable performance, specifically in regions where other models exhibit oscillations and sharp increases in error. One such region is clearly observed within the first 100 time steps, where most baseline models exhibit significant error spikes. In particular, during the interval between the 80th and 100th data points, the dependency of the previous observation y_{t-1} decreases. This leads to a significant decline in performance for models that overfit the auto-regressive features. In contrast, our model maintains stable performance and is not affected as sharply as the other models by this change.

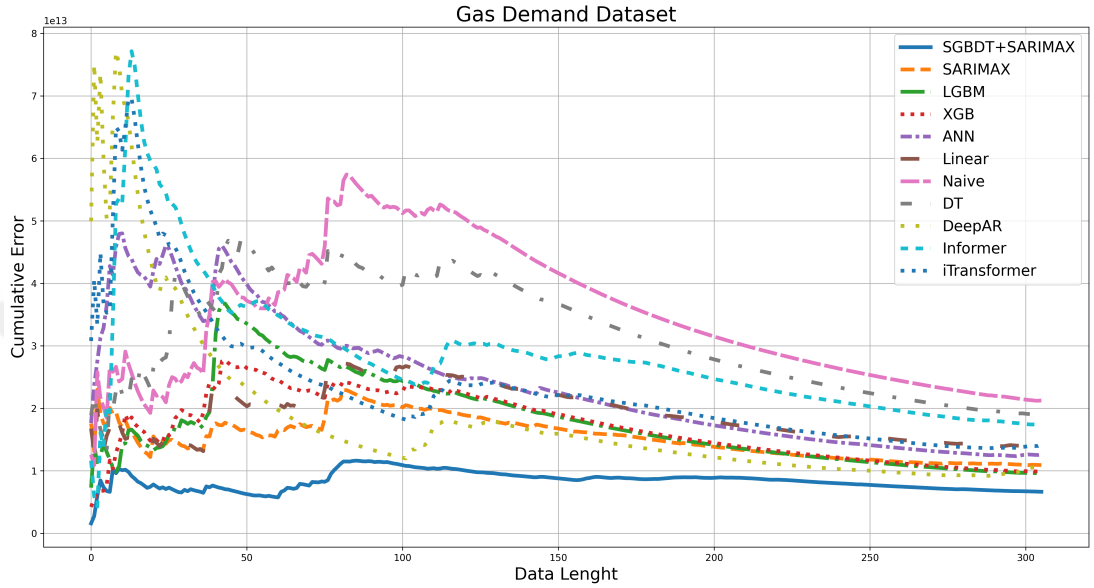

Figure 3.1: Comparison of cumulative mean squared error performance on Turkey Gas Demand Dataset

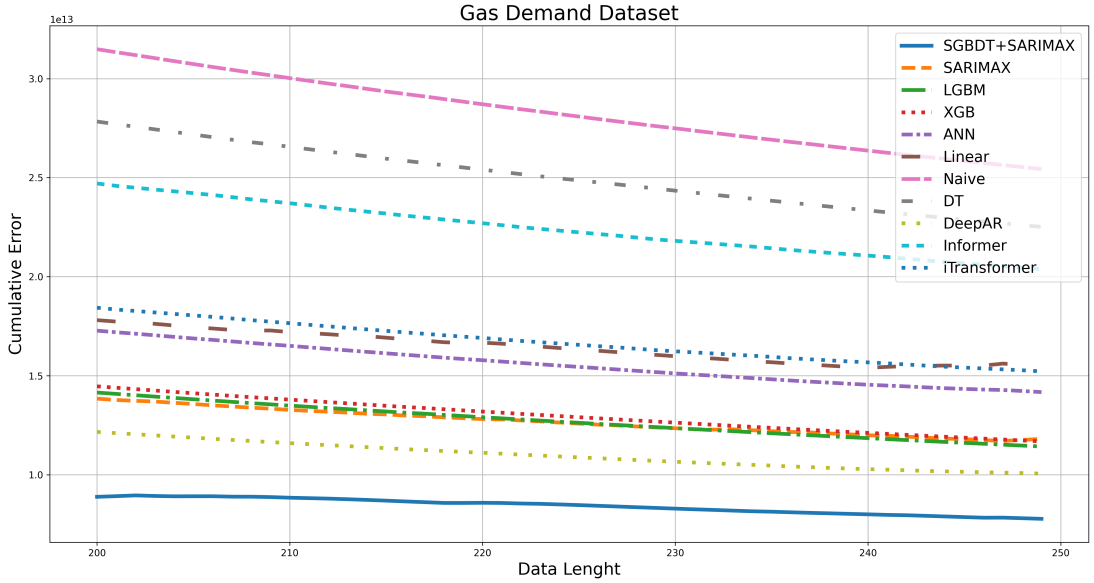

Figure 3.2: Zoomed in to comparison of cumulative mean squared error performance on Turkey Gas Demand Dataset to show differences in detail

	Elevators MSE(10^{-6})	Gas Demand MSE(10^{12})	Pumadyn MSE(10^{-5})
Naive	10.83	21.33	169.10
Linear Regressor	14.30	13.93	74.41
Decision Tree	17.02	18.97	11.08
SARIMAX	4.15	10.96	81.00
ANN	6.34	12.16	74.41
DeepAR	5.11	9.47	71.66
Informer	6.62	13.68	39.19
iTransformer	5.37	14.49	94.71
LightGBM	4.60	9.68	6.81
XGBoost	4.74	9.36	6.68
SGBDT + SARIMAX	**2.87**	**8.82**	**6.14**

Table 3.1: MSE performances of the models on real-life Datasets

3.2 M4 Competition Dataset

The M4 Competition Dataset is a widely recognized and publicly available benchmark used to evaluate time series forecasting models. It consists of 100,000 time series with varying lengths, frequencies, and characteristics. These sequences span multiple time resolutions, including hourly, daily, weekly, monthly, and yearly intervals. Notably, the dataset provides the data as training and test sets. Hence, we use the original split in our experiments without modification in training or test length. In our experimental setup, we select three specific time resolutions: hourly, daily, and weekly. The hourly subset consists of 414 individual time series, each exhibiting distinct lengths and scales. This dataset is particularly suited for analyzing short-term and capturing intraday fluctuations. Next, the daily dataset provides 4,227 time series, with an average sequence length of 2,357.4, indicating that the series are relatively long. This dataset enables us to evaluate the model's capacity to learn and generalize across long sequences while capturing typical patterns such as daily seasonality and weekday effects. Finally, we select the weekly dataset, including 359 time series. In contrast to the hourly dataset, this setting allows us to evaluate model performance in capturing long-term patterns, such as those associated with business cycles. The predefined forecast horizons for these datasets are 48, 14, and 13 time steps for the hourly, daily,

and weekly datasets, respectively. As a result, these different datasets allow us to comprehensively evaluate each model's capacity to capture temporal dynamics, from rapid intraday fluctuations to longer-term seasonal and cyclical patterns.

To ensure a fair and meaningful evaluation on the M4 Competition Dataset, we use the Mean Absolute Percentage Error (MAPE) as our prediction error metric. Since the dataset contains time series with widely varying scales, even within the same type. Metrics such as Mean Absolute Error (MAE) and Mean Squared Error (MSE) are not directly appropriate for the M4 competition dataset because of the varying scales. In contrast, the MAPE represents errors as relative percentages, ensuring scale-independence and making it a suitable metric given the variability in the M4 Competition Dataset, defined as:

$$\text{MAPE}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{1}{n} \sum_{i=1}^{n} \left| \frac{y_i - \hat{y}_i}{y_i} \right|, \tag{3.3}$$

where n represents the total number of test instances, y_i and $\hat{y}_i$ denote the target and predicted values of the ith instance, respectively.

In our experimental setup, we employ the same diverse set of baseline models to compare with our proposed model. These include XGBoost, LightGBM, iTransformer, Informer, DeepAR, ANN, SARIMAX, Decision Tree, Linear Regressor, and the Naive model. To enrich the input space and improve model performance, we extract a range of time series features such as rolling means, rolling standard deviations, and various lag values. The choice of lag intervals and rolling window sizes is adapted to each specific M4 dataset type, reflecting the temporal characteristics of the corresponding resolution. For example, in the hourly dataset, we include lag features at 1, 2, 3, 12, 24, 36, and 48-hour intervals so that we can use the information from the past day and up to two days prior. In contrast, for the weekly dataset, we select lags at 1, 2, 3, 4, and 52 weeks to capture both patterns in recent weeks, information from the past month, and seasonal trends on an annual scale. While tuning hyperparameters, we hold out a validation set from the end portion of the training data, and the size of the validation set is the same as the predefined size of the test set. The optimal hyperparameters are selected based on the best validation performance in terms of MAPE. To reduce

computational cost while maintaining representative diversity data, we randomly select 100 time series from each dataset type and compute the average MAPE score as shown in Table 3.2.

Table 3.2 presents the comparative results across the selected M4 Competition Datasets. It clearly shows that our proposed model, which jointly optimizes the SARIMAX and the SGBDT, consistently outperforms all other competing models. In the hourly dataset, SARIMAX and XGBoost exhibit comparable performance. Our ensemble model, which integrates the strengths of both the SARIMAX and the soft variant of gradient boosting trees, achieves lower prediction errors than either of these individual models. This highlights the benefits of their joint formulation. In the daily dataset, the effectiveness of the Naive model reveals a significant dependency on y_{t-1}, suggesting a strong autocorrelation in the data. As a result, simpler baseline models, namely the Naive and Linear Regressor, demonstrate competitive performance, particularly when compared to more complex methods such as LightGBM and XGBoost. In contrast, Informer and iTransformer, which are designed primarily to capture long-range temporal dependencies, perform less effectively in this dataset. Moreover, the DeepAR and the SARIMAX, both of which incorporate auto-regressive mechanisms, show significant improvements and outperform several SOTA models. Within our model, the SARIMAX component plays a prominent role, contributing to superior performance and yielding the lowest error among all compared models. In contrast, the weekly dataset has no strong dependency on y_{t-1}, but rather shows long-term temporal dependencies. As a result, transformer-based models perform better than other models. Nevertheless, our model slightly outperforms them and maintains its overall superiority.

3.3 Ablation Study

In this section, we present an ablation study to systematically analyze the contribution of each key component of our proposed model. The complete architecture consists of three unique aspects: (i) state-space formulations of both SGBDT

	Hourly	Daily	Weekly
Naive	0.06339	0.01604	0.20122
Linear Regressor	0.02752	0.01643	0.07231
Decision Tree	0.03577	0.02314	0.09881
SARIMAX	0.03123	0.01652	0.15152
ANN	0.08535	0.02219	0.09198
DeepAR	0.75225	0.01549	0.11311
Informer	0.08088	0.01866	0.06003
iTransformer	0.07664	0.02713	0.05911
LightGBM	0.02606	0.01944	0.07962
XGBoost	0.03129	0.02050	0.07434
SGBDT + SARIMAX	**0.02024**	**0.01413**	**0.05658**

Table 3.2: Mean MAPE performances of the models on M4 Competition Datasets

and SARIMAX, (ii) their ensemble within the state-space domain, and (iii) a joint optimization by using EKF. To evaluate the effect of each component in isolation, we conduct a series of step-by-step comparisons. The comparisons begin with an evaluation of the individual performance of the two core components: the standalone SARIMAX model and an SDT trained via stochastic gradient descent (SGD). In the next step, we add the gradient boosting algorithm to transition from SDT to SGBDT so that we can analyze the effect of boosting. Finally, to isolate the impact of joint optimization, we construct an ensemble of SGBDT and SARIMAX trained independently, i.e., using a disjoint training strategy. This enables us to compare disjoint and joint optimization. We conduct the experiments on the Bank Dataset [53], which is a simulation-based dataset modeling customer behavior in the context of bank selection. The prediction target is the fraction of customers who leave a bank due to full queues. Each instance represents customer decision-making based on factors such as distance, perceived complexity, and individual patience levels. The dataset consists of 4,500 samples and 32 features. No additional feature engineering is performed. We allocate 20% of the data for testing, 20% for validation, and use the remaining for training. In Fig. 3.3, we illustrate the $CumMSE$ for each model, computed as defined in 3.2. According to Fig. 3.3, the gradient boosting algorithm significantly reduces CumMSE in the Bank Dataset, and the SARIMAX outperforms SDT in terms of cumulative error. On the other hand, the incorporation of the gradient boosting

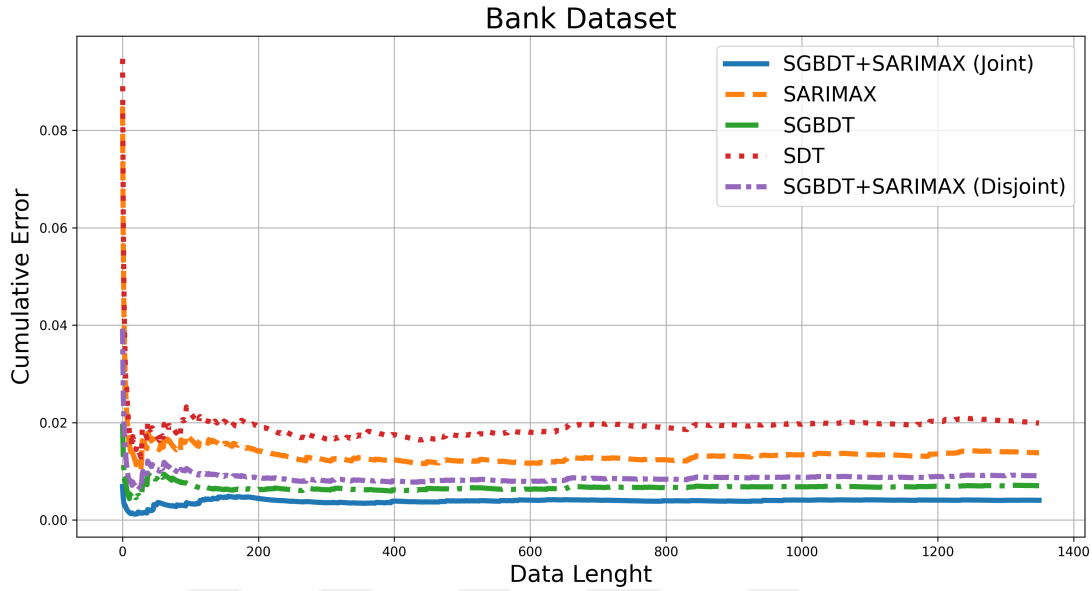

Figure 3.3: Ablation Study of cumulative mean squared error performance on Bank Dataset

algorithm enables SGBDT to outperform SARIMAX. Next, the model trained under a disjoint optimization strategy demonstrates sub-optimal performance, i.e., its performance is similar to a simple weighted average of the standalone SARIMAX and SGBDT models. As a result, our proposed model consistently achieves the lowest cumulative error across the entire test dataset, indicating its robustness and effectiveness. This performance improvement is attributed to the joint optimization of the SGBDT and SARIMAX components, which outperforms both the individual models and the disjointly optimized ensemble.

3.4 5G Application

In this section, we work on a 5G Traffic forecasting problem. We make a comprehensive study on a publicly available dataset [54]. The dataset is collected from a real commercial 5G network in South Korea using a 5G-enabled mobile terminal. Measurements are conducted at a fixed location under stable network conditions to ensure consistency and avoid traffic fluctuations caused by peak usage hours.

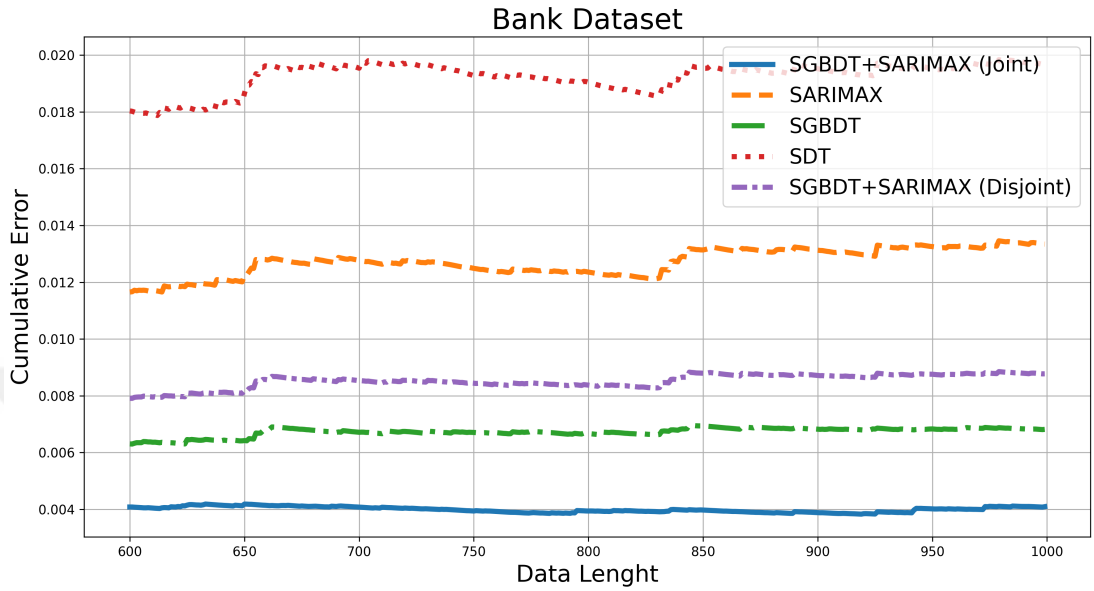

Figure 3.4: Zoomed in to ablation study of cumulative mean squared error performance on Bank Dataset to show the differences in detail

The dataset contains packet-level traffic traces from various popular applications.

To analyze the dataset, we first conduct an EDA. We chose the YouTube-Live data. The sampled data is recorded in datetime format with millisecond-level granularity and spans from 08:00 to 12:00 on May 30, 2022. It consists of 914,413 data instances. Each entry includes the date, source IP, destination IP, and packet length, as the target variable for forecasting. We aggregate the packet lengths over time to convert the resolution into seconds, resulting in a transformed time series as shown in Fig. 3.5.

In time series forecasting, stationarity is a critical assumption for many statistical models, including ARIMA-based models. Since the SARIMAX model is used in both our proposed model and the baseline models for comparison, it is essential to test the stationarity of the time series. Hence, we need to verify whether the series is stationary or not. For this purpose, we apply the Augmented Dickey-Fuller (ADF) test, which assesses the null hypothesis that a unit root is present in the time series (i.e., the series is non-stationary). We obtain the p-value as 1.26e-22, which is far below the typical threshold of 0.05. Therefore, we reject the null hypothesis and conclude that the series is stationary. Accordingly,

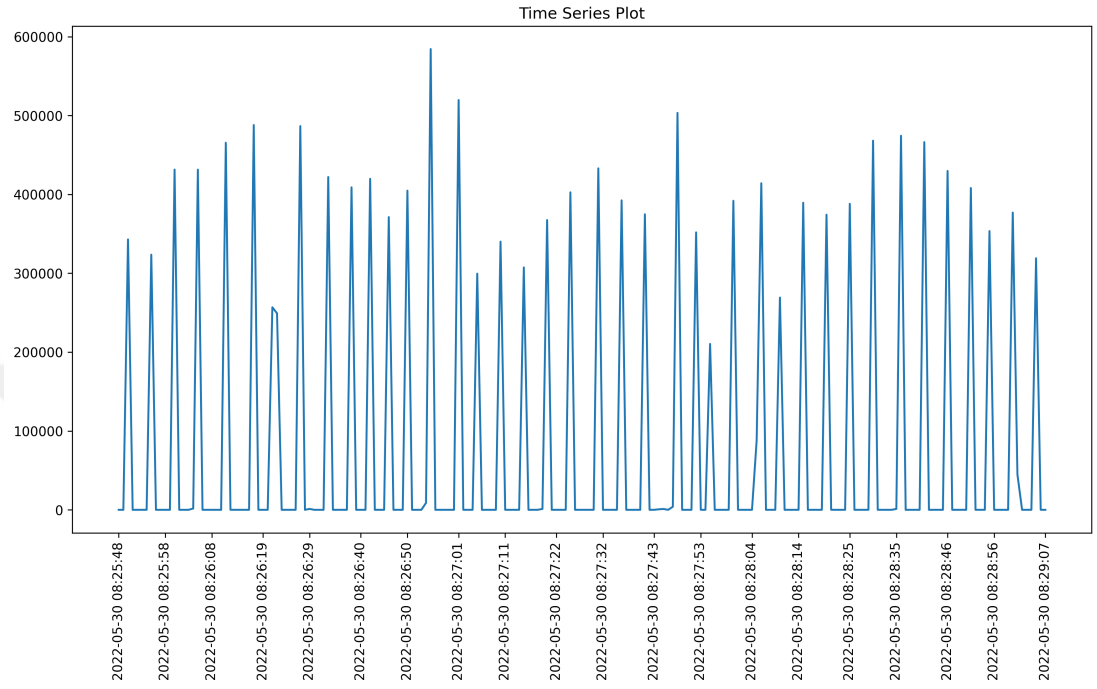

Figure 3.5: First 200 samples for the 5G Youtube-Live Dataset.

there is no need for a differencing operation in ARIMA-based models; hence, we can set the differencing order to 0.

We extract lags and rolling statistics (mean and standard deviation). To select appropriate lag values, we first analyze the Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF). The ACF measures the correlation between a time series and its lagged values and identifies the MA order (q) in ARMA-based models, whereas the PACF measures correlation between current and lagged values by isolating the direct correlation between a time series and its lags and identifies the AR order (p). In the ACF plot of Fig. 3.6, it shows a slow decay rather than a sharp cutoff. Therefore, the series has a persistent autocorrelation structure. This pattern suggests the presence of an auto-regressive (AR) process rather than a moving average (MA) process. This enables us to narrow the MA order search space. On the other hand, the PACF plot shows significant spikes up to lag 4, and after lag 4, the values become small and oscillate around zero. This sharp cutoff implies that an AR order of 4 (i.e., $p = 4$) is appropriate for this series. As a result, we need to certainly include 4 in the hyperparameter

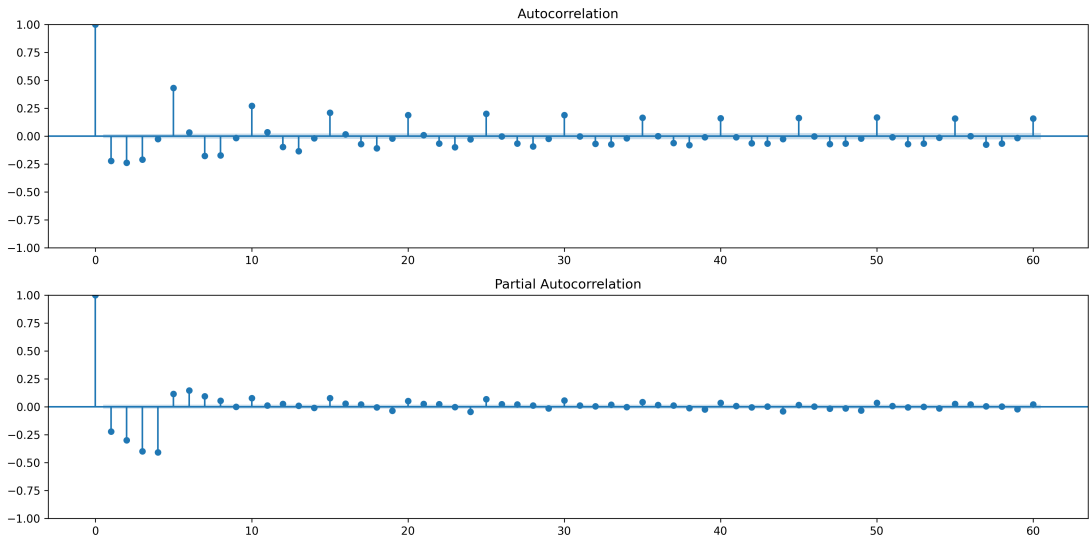

Figure 3.6: ACF and PACF plots of the 5G Youtube-Live series.

search space of the AR order and prioritize configurations with $p = 4$ for the SARIMAX model and our proposed model. For other models, we need to extract lag features at lags 1 through 4 and a rolling mean with a window length of 4 to capture short-term memory.

Next, we extract time-related features such as hours, minutes, and seconds. First, we analyze the average packet length per hour throughout the day. Since our dataset spans a single day, this analysis allows us to determine which hours correspond to the most and least activity. As shown in the heat map in Fig. 3.7, the highest activity is observed at 9:00 AM, while the lowest occurs at 10:00 AM.

In Fig. 3.8, the heatmap visualizes average packet length calculated for each second within every minute. This heatmap reveals clear burst patterns, with specific seconds (e.g., 5, 25, 50) showing consistently higher average values across multiple minutes, which suggests periodic or event-driven activity. Therefore, we extract the sine and cosine values of the seconds to obtain cyclic encoding based on the second. Cyclic features make this burst pattern learnable for our models.

Similar to the Real-Life Dataset (3.1), we allocate 30% of the entire data for test and use the remaining 70% for training. We use hold-out validation from the last 30% of the training data. Before starting the test, we refit models to the

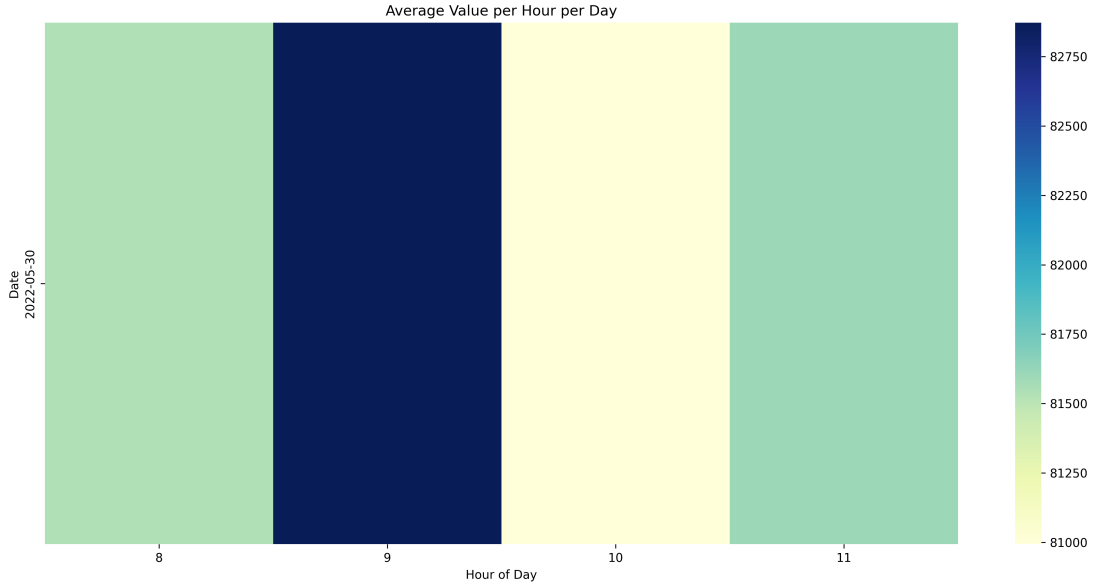

Figure 3.7: Temporal heatmap representing average packet lengths by hour on May 30, 2022.

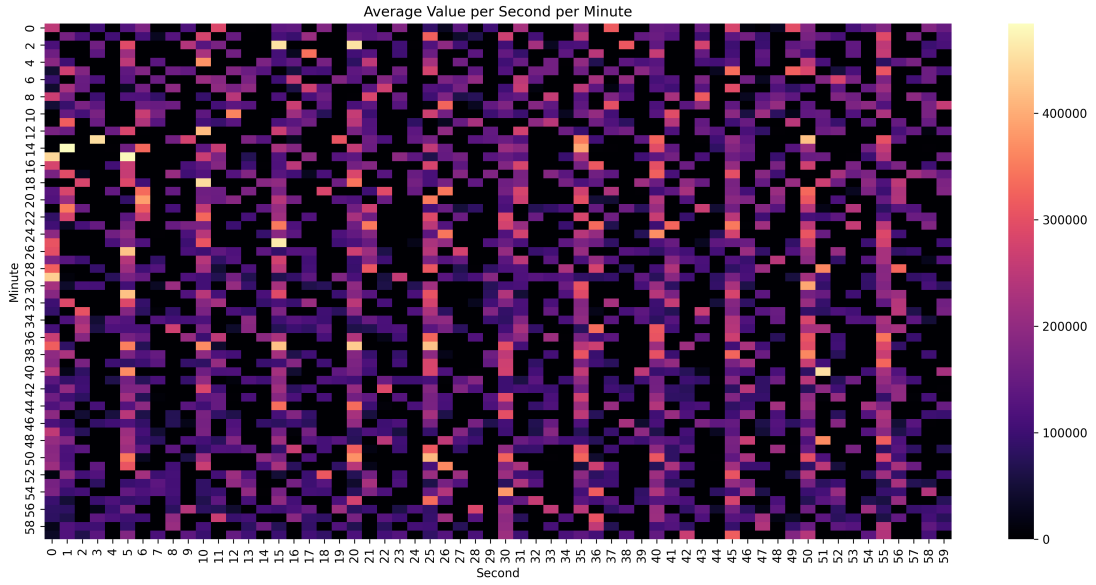

Figure 3.8: Temporal heatmap representing average packet lengths per second across each minute.

entire training data. We apply standard normalization during preprocessing for models that require it, such as neural networks. Additionally, we perform post-processing on all model outputs by clipping negative predictions to zero since the target variable cannot take negative values. We further evaluate model performance by using both MSE and MAE as evaluation metrics. In Table 3.3, we present the error performance of all models on the test. Based on the Naive model performance, we can understand that the data exhibits no significant dependency on y_{t-1}. Informer, LightGBM, and XGBoost show competitive results with MSE values of 1.436×10^{10}, 1.463×10^{10}, and 1.487×10^{10}, respectively. Traditional neural models such as GRU, LSTM, and RNN fail to perform adequately compared to tree-based models and SOTA deep learning models, with MSEs ranging from 2.091×10^{10} to 2.229×10^{10}, while MAEs were close to 9.5×10^4. These results indicate that they struggle to generalize well to fluctuations in 5G traffic. Among the deep temporal models, DeepAR, Informer, and iTransformer perform better. That is, Informer achieves the lowest MSE of 1.436×10^{10}, and DeepAR shows the best MAE among them at 7.204×10^4. Tree-based models such as XGBoost and LightGBM surpass the deep models that both achieve lower MAE values around 6.5×10^4. The SARIMAX, linear regressor, and decision tree demonstrate reasonable predictive performance, with MSE values of 1.780×10^{10}, 1.696×10^{10}, and 1.683×10^{10}, respectively. SVR performs worse than these baselines in terms of MSE, but its lower MAE indicates that it may produce large outliers. The disjointly optimized SGBDT + SARIMAX shows comparable performance to other top-tier models. Then, our proposed jointly optimized SGBDT + SARIMAX model achieves the best performance, substantially outperforming all compared models with an MSE of 1.240×10^{10} and an MAE of 3.411×10^4. Hence, compared to disjoint optimization, this significant reduction in both squared and absolute error metrics highlights the effectiveness of jointly integrating statistical and machine learning components to capture both temporal dynamics and nonlinear feature interactions.

In Fig. 3.9, our model demonstrates the best overall performance in terms of CumMSE, maintaining the lowest cumulative error throughout the entire test

	MSE(10^{10})	MAE(10^4)
Naive	6.814	15.904
Linear Regressor	1.696	9.356
SVR	3.459	8.120
SARIMAX	1.780	9.756
Decision Tree	1.683	6.530
ANN	1.625	9.043
RNN	2.229	9.824
GRU	2.097	9.499
LSTM	2.091	9.597
DeepAR	1.555	7.204
Informer	1.436	7.230
iTransformer	1.538	7.947
XGBoost	1.487	6.623
LightGBM	1.463	6.466
SGBDT + SARIMAX (Disjoint)	1.539	8.342
SGBDT + SARIMAX (Joint)	**1.240**	**3.411**

Table 3.3: MSE and MAE performances of the models on 5G Youtube-Live Dataset

set. The ensemble approach of SGBDT + SARIMAX (Disjoint) shows a noticeable improvement over individual models, but it underperforms compared to the SGBDT + SARIMAX (Joint) model. Our model benefits from a unified architecture and achieves a sharp reduction in cumulative error. We also provide the zoomed in version in Fig. 3.10 to distinguish models with very close error patterns.

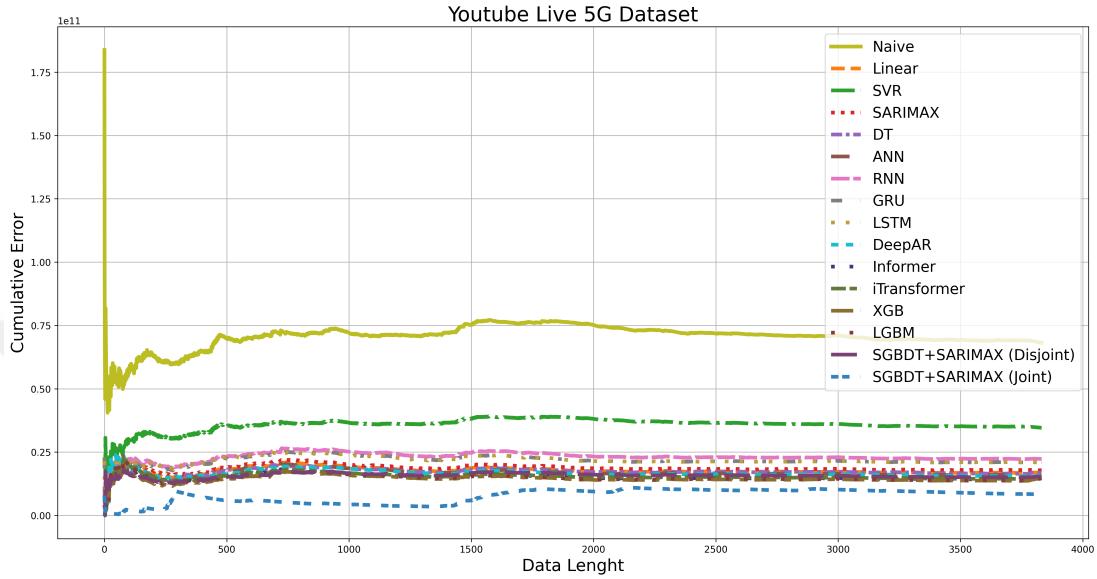

Figure 3.9: Cumulative mean squared error performance on 5G Youtube-Live Dataset

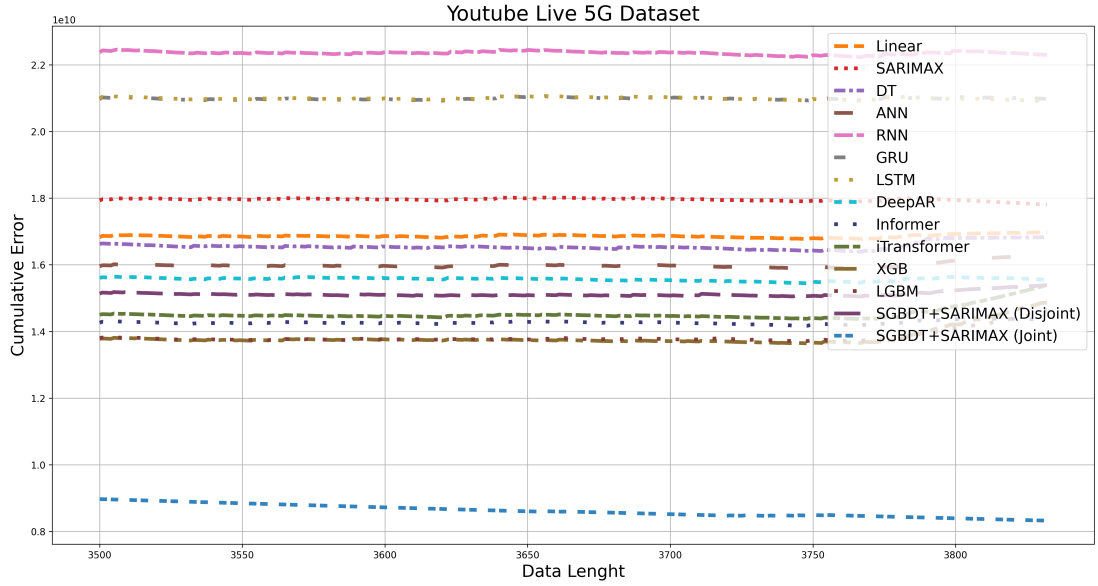

Figure 3.10: Zoomed in to cumulative mean squared error performance on 5G Youtube-Live Dataset to show the differences in detail. The Naive and SVR models are excluded due to their substantially higher error curves, which distort comparative analysis.

Chapter 4

Conclusion

In this research, we address the challenge of nonlinear prediction in an online setting. We present a novel unified architecture that integrates SGBDT (Soft Gradient Boosting Decision Trees) with SARIMAX (Seasonal Autoregressive Integrated Moving Average with Exogenous Variables). Different from the common practice of employing disjoint ensemble models, our approach unifies the two models within a single state-space formulation. Then, this single formulation enables us to combine nonlinear predictive power and differentiability of SGBDTs with the robust linear model SARIMAX to handle time series-specific features such as seasonality. By eliminating the traditional separation between domain-specific feature selection and model training, we introduce a more efficient hybrid approach that leverages both nonlinear and linear components, while maintaining the benefits of end-to-end optimization. We train the ensemble model through EKF and present the detailed learning equations for the model parameters. One of the main contributions of this work is the adaptability of our architecture, which supports a range of differentiable tree models, including other gradient boosting decision trees as well as conventional models, namely ETS. We explain our comprehensive experimental setup in detail. Additionally, we provide the complete feature engineering and fair hyperparameter tuning process. Our experiments demonstrate consistent and significant performance improvements over statistical and machine learning models, including state-of-the-art methods, on

multiple well-known datasets and a 5G application. These experimental results demonstrate the effectiveness of the proposed approach in capturing both linear and nonlinear patterns in time series data. We also provide the source code for our model to facilitate further research and reproducibility of our results.

Bibliography

[1] Y. Zhang, "Hourly traffic forecasts using interacting multiple model (imm) predictor," *IEEE Signal Processing Letters*, vol. 18, no. 10, pp. 607–610, 2011.

[2] M. Xu, Y. Yang, M. Han, T. Qiu, and H. Lin, "Spatio-temporal interpolated echo state network for meteorological series prediction," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 6, pp. 1621–1634, 2019.

[3] L. Zhang, G. Wang, and G. B. Giannakis, "Real-time power system state estimation and forecasting via deep unrolled neural networks," *IEEE Transactions on Signal Processing*, vol. 67, no. 15, pp. 4069–4077, 2019.

[4] G. Dudek, P. Pełka, and S. Smyl, "A hybrid residual dilated lstm and exponential smoothing model for midterm electric load forecasting," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 7, pp. 2879–2891, 2022.

[5] T. Vercauteren, P. Aggarwal, X. Wang, and T.-H. Li, "Hierarchical forecasting of web server workload using sequential monte carlo training," *IEEE Transactions on Signal Processing*, vol. 55, no. 4, pp. 1286–1297, 2007.

[6] Z. Zhang, S. Zohren, and S. Roberts, "Deeplob: Deep convolutional neural networks for limit order books," *IEEE Transactions on Signal Processing*, vol. 67, no. 11, pp. 3001–3012, 2019.

[7] A. Singer, G. Wornell, and A. Oppenheim, "Nonlinear autoregressive modeling and estimation in the presence of noise," *Digital Signal Processing: A Review Journal*, vol. 4, pp. 207–221, Oct. 1994.

[8] F. Petropoulos et al., "Forecasting: theory and practice," *International Journal of Forecasting*, Jan 2022.

[9] L. Liu, L. Shao, and P. Rockett, "Human action recognition based on boosted feature selection and naive bayes nearest-neighbor classification," *Signal Processing*, vol. 93, no. 6, pp. 1521–1530, 2013. Special issue on Machine Learning in Intelligent Image Processing.

[10] X. An, C. Hu, G. Liu, and H. Lin, "Distributed online gradient boosting on data stream over multi-agent networks," *Signal Processing*, vol. 189, p. 108253, 2021.

[11] M. E. Aydin and S. S. Kozat, "A hybrid framework for sequential data prediction with end-to-end optimization," *Digital Signal Processing*, vol. 129, p. 103687, 2022.

[12] H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang, "Informer: Beyond efficient transformer for long sequence time-series forecasting," *CoRR*, vol. abs/2012.07436, 2020.

[13] Y. Liu, T. Hu, H. Zhang, H. Wu, S. Wang, L. Ma, and M. Long, "itransformer: Inverted transformers are effective for time series forecasting," 2024.

[14] T. Zhou, Z. Ma, Q. Wen, X. Wang, L. Sun, and R. Jin, "Fedformer: Frequency enhanced decomposed transformer for long-term series forecasting," *CoRR*, vol. abs/2201.12740, 2022.

[15] S.-A. Chen, C.-L. Li, S. Arik, N. C. Yoder, and T. Pfister, "Tsmixer: An all-mlp architecture for time series forecast-ing.," *Trans. Mach. Learn. Res.*, vol. 2023, 2023.

[16] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," in *Advances in Neural Information Processing Systems* (I. Guyon, U. V. Luxburg,

S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, Curran Associates, Inc., 2017.

[17] E. S. Gardner Jr., "Exponential smoothing: The state of the art," *Journal of Forecasting*, vol. 4, no. 1, pp. 1–28, 1985.

[18] J. Feng, Y. Xu, Y. Jiang, and Z. Zhou, "Soft gradient boosting machine," *CoRR*, vol. abs/2006.04059, 2020.

[19] O. İrsoy, O. T. Yıldız, and E. Alpaydın, "Soft decision trees," in *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, pp. 1819–1822, 2012.

[20] E. K. Sahin, "Assessing the predictive capability of ensemble tree methods for landslide susceptibility mapping using xgboost, gradient boosting machine, and random forest," *SN Applied Sciences*, vol. 2, no. 7, p. 1308, 2020.

[21] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, (New York, NY, USA), pp. 785–794, ACM, 2016.

[22] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, "Catboost: unbiased boosting with categorical features," in *Advances in Neural Information Processing Systems* (S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, eds.), vol. 31, Curran Associates, Inc., 2018.

[23] C.-J. Lu, T.-S. Lee, and C.-C. Chiu, "Financial time series forecasting using independent component analysis and support vector regression," *Decision Support Systems*, vol. 47, no. 2, pp. 115–125, 2009.

[24] S. Dittmer, E. J. King, and P. Maass, "Singular values for relu layers," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 9, pp. 3594–3605, 2020.

[25] G. Li and J. Ding, "Towards understanding variation-constrained deep neural networks," *IEEE Transactions on Signal Processing*, vol. 71, pp. 631–640, 2023.

[26] W. Yan, "Toward automatic time-series forecasting using neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 23, no. 7, pp. 1028–1039, 2012.

[27] A. Sherstinsky, "Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network," *Physica D: Nonlinear Phenomena*, vol. 404, p. 132306, 2020.

[28] D. Salinas, V. Flunkert, J. Gasthaus, and T. Januschowski, "Deepar: Probabilistic forecasting with autoregressive recurrent networks," *International Journal of Forecasting*, vol. 36, no. 3, pp. 1181–1191, 2020.

[29] H. Wu, J. Xu, J. Wang, and M. Long, "Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting," *CoRR*, vol. abs/2106.13008, 2021.

[30] S. Wang, H. Wu, X. Shi, T. Hu, H. Luo, L. Ma, J. Y. Zhang, and J. ZHOU, "Timemixer: Decomposable multiscale mixing for time series forecasting," in *The Twelfth International Conference on Learning Representations*, 2024.

[31] B. Lim, S. Arık, N. Loeff, and T. Pfister, "Temporal fusion transformers for interpretable multi-horizon time series forecasting," *International Journal of Forecasting*, vol. 37, no. 4, pp. 1748–1764, 2021.

[32] M. Khodarahmi and V. Maihami, "A review on kalman filter models," *Archives of Computational Methods in Engineering*, vol. 30, no. 2, pp. 727–747, 2023.

[33] C. Urrea and R. Agramonte, "Kalman filter: Historical overview and review of its use in robotics 60 years after its creation," *Journal of Sensors*, vol. 2021, pp. Article ID 9674015, 21 pp., 2021. Open Access, CCBY 4.0.

[34] A. Joshuva and V. Sugumaran, "A machine learning approach for condition monitoring of wind turbine blade using autoregressive moving average

(arma) features through vibration signals: a comparative study," *Progress in Industrial Ecology, an International Journal*, vol. 12, no. 1-2, pp. 14–34, 2018.

[35] C. Li, X. Zheng, Z. Yang, and L. Kuang, "Predicting short-term electricity demand by combining the advantages of arma and xgboost in fog computing environment," *Wireless Communications and Mobile Computing*, vol. 2018, no. 1, p. 5018053, 2018.

[36] L. Yang, J. Zhang, and X. Yang, "Prediction of the reported results of wordle based on arima and gbdt," in *2023 International Conference on Computers, Information Processing and Advanced Education (CIPAE)*, pp. 108–115, 2023.

[37] J. Farajzadeh and F. Alizadeh, "A hybrid linear–nonlinear approach to predict the monthly rainfall over the Urmia Lake watershed using wavelet-SARIMAX-LSSVM conjugated model," *Journal of Hydroinformatics*, vol. 20, pp. 246–262, 08 2017.

[38] J. Durbin and S. J. Koopman, *Time Series Analysis by State Space Methods*. Oxford University Press, 05 2012.

[39] H. Li, J. Song, M. Xue, H. Zhang, and M. Song, "A survey of neural trees: Co-evolving neural networks and decision trees," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–20, 2024.

[40] N. S. Arunraj, D. Ahrens, and M. Fernandes, "Application of sarimax model to forecast daily sales in food retail industry," *International Journal of Operations Research and Information Systems (IJORIS)*, vol. 7, no. 2, pp. 1–21, 2016.

[41] R. J. Hyndman and G. Athanasopoulos, *Forecasting: principles and practice*. OTexts, 2018.

[42] R. Zhao, W. B. Haskell, and V. Y. F. Tan, "Stochastic l-bfgs: Improved convergence rates and practical acceleration strategies," *IEEE Transactions on Signal Processing*, vol. 66, no. 5, pp. 1155–1169, 2018.

[43] T. Wigren, "Fast converging and low complexity adaptive filtering using an averaged kalman filter," *IEEE Transactions on Signal Processing*, vol. 46, no. 2, pp. 515–518, 1998.

[44] X. Wang and Y. Huang, "Convergence study in extended kalman filter-based training of recurrent neural networks," *IEEE Transactions on Neural Networks*, vol. 22, no. 4, pp. 588–600, 2011.

[45] K. Reif and R. Unbehauen, "The extended kalman filter as an exponential observer for nonlinear systems," *IEEE Transactions on Signal Processing*, vol. 47, no. 8, pp. 2324–2328, 1999.

[46] G. Einicke and L. White, "Robust extended kalman filtering," *IEEE Transactions on Signal Processing*, vol. 47, no. 9, pp. 2596–2599, 1999.

[47] T. De Ryck, M. De Vos, and A. Bertrand, "Change point detection in time series data using autoencoders with a time-invariant representation," *IEEE Transactions on Signal Processing*, vol. 69, pp. 3513–3524, 2021.

[48] S. Makridakis, E. Spiliotis, and V. Assimakopoulos, "The m4 competition: 100,000 time series and 61 forecasting methods," *International Journal of Forecasting*, vol. 36, no. 1, pp. 54–74, 2020. M4 Competition.

[49] J. Alcala-Fdez, A. Fernández, J. Luengo, J. Derrac, S. Garc'ia, L. Sanchez, and F. Herrera, "Keel data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework," *Journal of Multiple-Valued Logic and Soft Computing*, vol. 17, pp. 255–287, 01 2010.

[50] D. C. Montgomery, E. A. Peck, and G. G. Vining, *Introduction to Linear Regression Analysis (4th ed.)*. Wiley & Sons, 2006.

[51] V. Borisov, T. Leemann, K. Seßler, J. Haug, M. Pawelczyk, and G. Kasneci, "Deep neural networks and tabular data: A survey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 35, no. 6, pp. 7499–7519, 2024.

[52] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations (ICLR)*, (San Diega, CA, USA), 2015.

[53] L. Torgo, "Regression data sets." https://www.dcc.fc.up.pt/~ltorgo/Regression/DataSets.html.

[54] D. Kim, "5g traffic datasets." https://www.kaggle.com/datasets/kimdaegyeom/5g-traffic-datasets, 2021.